Dynamic Aspects of Character Rendering in the Context of Multimodal Dialog Systems

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

von
Dipl.-Inform. (FH) Yvonne A. Jung
geb. in Neuwied

Referenten der Arbeit:	Prof. Dr. Dieter W. Fellner
	Prof. Dr. Didier Stricker
Tag der Einreichung:	23.12.2010
Tag der mündlichen Prüfung:	07.02.2011

D17
Darmstadt 2011

Selbständigkeitserklärung

Hiermit versichere ich, die vorliegende Dissertation selbständig und ausschließlich unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 23. Dezember 2010

Yvonne Jung

Acknowledgment

First of all, I'd like to express my gratitude to my PhD supervisor Prof. Dr. Dieter Fellner for his support. I'd also like to thank PD Dr. Arjan Kuijper for his assistance and advice when writing this thesis and Dr. Johannes Behr for fruitful discussions.

In addition, I would also like to thank all my colleagues from the Department for Virtual and Augmented Reality at the Fraunhofer IGD in Darmstadt for their support and friendship as well as my students for their help, especially Christine who also provided the hair photos on page 135.

Finally, I'd like to express my sincere gratitude to my parents, my sister, and my grandfather for their motivating encouragement as well as to my ponies for being there.

Abstract

For many application areas, where a task is most naturally represented by talking or where standard input devices are difficult to use or not available at all, virtual characters can be well suited as an intuitive man-machine-interface due to their inherent ability to simulate verbal as well as nonverbal communicative behavior. This type of interface is made possible with the help of multimodal dialog systems, which extend common speech dialog systems, as for instance known from automated technical support hotlines, with additional modalities just like in human-human interaction. Such a dialog system consists at least of an auditive and graphical component and, comparable to face-to-face dialogs, communication is based on speech and nonverbal communication alike.

However, employing virtual characters as personal and believable dialog partners in multimodal dialogs entails several challenges, because this not only requires a reliable and consistent motion and dialog behavior, which also regards nonverbal communication and affective components, but also efficiency for the realization of applications. Besides modeling the "mind" and creating intelligent communication behavior on the encoding side, which is an active field of research in artificial intelligence, the visual representation of a character including its perceivable behavior (from a decoding perspective), such as facial expressions and gestures, belongs to the domain of computer graphics and likewise implicates many open issues concerning natural communication.

Although there already exist many systems to simulate virtual characters, they are mostly focused on certain subdomains, designed as standalone application using proprietary formats and in-house libraries, or they do not address the demands of interactive and dynamic environments as particularly given in Mixed and Virtual Reality environments. Also, the prospects of advanced real-time rendering techniques as known from e.g. computer games still are mostly ignored in the context of embodied conversational agents research. In addition, the embodied agent as a user interface component needs to be tightly integrated into larger applications to allow for interactions between the virtual character itself, the user, and the 3D world. Moreover, embedding the character is also necessary to avoid missing contextual relevance or interactions with the agent that appear artificial.

Therefore, this work mainly focuses on the definition and development of a generic system for the visualization component of multimodal dialog systems, which integrates relevant functionalities and provides them in a manageable manner. In this regard, the Instant Reality framework is a suitable visualization platform in that it enables a broad range of interactive Virtual and Augmented Reality applications, because it already provides a multitude of multimodal input and output interfaces, which eases the interactions between real and virtual humans. Availability and efficiency are further guaranteed by integrating the proposed techniques into established standards like the scene-graph and into X3D, which is currently the only standardized 3D deployment format.

Besides this, related subquestions such as camera and animation control are considered, too, since a really interactive virtual character requires a high degree of control over its body, face, voice, and its physiological or externally observable processes in general, whereas the system must be able to emphasize or clarify any of these with suitable camera work. High-level control and flexibility is achieved by introducing another level of abstraction on top of the open ISO standard X3D by means of a higher level interface and control language, which can be used for module communication and for coordinating and synchronizing the conversational behavior of virtual humans.

However, the focus clearly lies on the graphical representation of virtual characters, while especially focusing on the dynamic aspects of rendering. Therefore, a self-contained and integrated system with matching techniques and building blocks for rendering and animation is proposed, that not only provides flexible control of the character concerning gestures, mimics, and speech, but also considers resultant dependencies, which need to be simulated during runtime such as long hair blowing in the wind or tears dripping down.

Furthermore, the system also takes psycho-physiological processes such as blushing and crying into account. These cannot be controlled deliberately, but need to be consistent and synchronous with the character's motor response. Though these effects are mostly ignored in research, they are essential for the correct perception of strong emotions in the context of nonverbal communication. Additionally, in case the agent shall be part of a Mixed Reality application, suitable real-time rendering methods are developed that permit a seamless integration of the virtual character with the remaining real scene.

Thus, the proposed approach offers sustainability and more efficiency by means of the integration into well established visualization techniques like the scene-graph, into existing open standards like X3D, and into the more abstract service-oriented system architectures as typically used in the embodied agents community. Finally, the applicability of the outlined concepts was proven in the course of various research and industry projects demonstrating different fields of applications. The presented system hence has the capability of being an enabling technology for the proliferation of multimodal dialog systems.

Zusammenfassung

Virtuelle Charaktere erfahren derzeit eine immer stärker werdende Bedeutung in den verschiedensten Bereichen, angefangen bei Filmen und Computerspielen über Infotainment und Trainingssimulatoren bis hin zu Online-Communities wie Second Life. Durch ihr natürliches Verhalten und Aussehen bieten geeignet gestaltete virtuelle Charaktere als Benutzerschnittstelle ein großes Potential, da sie es erlauben, kommunikatives Verhalten nachzubilden und zu simulieren. Einige Beispiele sind in Abbildung 0.1 gezeigt.

Kommunikation selbst wird dabei in der Pragmatik als Sprachhandeln aufgefasst, wozu neben rein textuell formulierten Äußerungen auch die nonverbale Kommunikation gehört, welche als Teil menschlichen Verhaltens immer stattfindet [334]. So kann nonverbales Verhalten eine Aussage unterstützen, etwa durch eine deiktische Geste, oder im Extremfall die Aussage sogar negieren. Im Gegensatz zu artifiziellen und zunächst zu erlernenden Benutzerschnittstellen stellt dieses Sprachhandeln meist die natürlichste Form der Kommunikation dar, da der Mensch von Kind an darauf trainiert ist, entsprechende Körpersignale zu identifizieren, zu interpretieren und sich dadurch mit seiner Umgebung zu verständigen.

Im Bereich der Computergraphik und der Künstlichen Intelligenz gibt es daher inzwischen zahlreiche Arbeiten, die sich im Bereich virtueller Charaktere mit den verschiedensten Aspekten befassen, wie beispielsweise der Simulation von Kleidern und Haaren, der realistischen Darstellung von Haut, unterschiedlichen Animationsmethoden oder auch generell von autonomem Verhalten. Doch trotz der rasanten Entwicklungsfortschritte der letzten Jahre aufgrund der den Markt treibenden Film- und Spieleproduktionen ist die Thematik immer noch ein weites und sehr aktives Forschungsfeld, angefangen bei der 3D-Modellerstellung selbst bis hin zur Modellierung des kommunikativen Verhaltens.

Moderne Computerspiele, technische Demonstrationen, Filmproduktionen und die aktuelle Literatur zeigen zwar die prinzipielle Machbarkeit, doch "mal schnell" einen interaktiven Avatar mit passendem Aussehen und Verhalten in eine beliebige Anwendung zu integrieren, ist noch immer schwierig bis unmöglich. Ein großes Problem ist auch, dass die verwendeten Verfahren oftmals nicht auf andere Applikationen bzw. Systeme übertragbar und somit von einer generellen Standardisierung noch weit entfernt sind. Das momentan sicherlich berühmteste Beispiel ist James Camerons "Avatar", ein Kinofilm, der einen sehr großen Anteil an photorealistischen, computergenerierten Charakteren enthält, der in der Erstellung jedoch auch extrem aufwendig und teuer war.

Multimodale Dialogsysteme

Wie eingangs erwähnt, können virtuelle Charaktere eine ideale Mensch-Maschine-Schnittstelle darstellen, vorausgesetzt, entsprechende Funktionalitäten werden bereitgestellt. Sol-

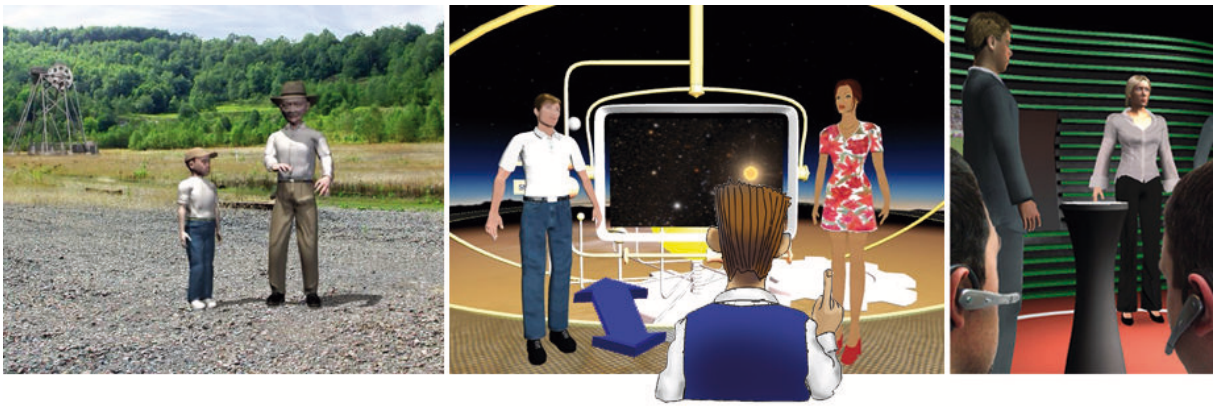

Abbildung 0.1: *Drei Beispielanwendungen aus dem Infotainmentbereich, bei denen sich virtuelle und echte Menschen miteinander unterhalten. Links: Vorgerenderte AR-Anwendung, bei der zwei virtuelle Charaktere im Dialog miteinander die Entstehungsgeschichte einer historischen Stätte erläutern. Mitte: Im Rahmen des Virtual Human Projekts [321] entstandener Demonstrator, bei dem ein menschlicher Benutzer (stilisiert dargestellt) im Dialog mit virtuellen Charakteren eine Unterrichtsstunde erhält. Rechts: Das interaktive Quizspiel ZAMB [321], bei dem man auch über Spracheingabe mit den Charakteren kommunizieren kann.*

che neuen Interaktionsmethoden werden mit der Entwicklung stets leistungsfähigerer und dabei allgegenwärtiger werdender Rechner einerseits und dem starken Zuwachs der zu verarbeitenden Datenmenge andererseits immer wichtiger, da sich die Aufnahmefähigkeit des Menschen selbst kaum verändert und zudem neue Benutzergruppen dazukommen. Gerade auch für unerfahrenere Benutzer oder für Anwendungsfälle, die am natürlichsten durch ein Gespräch repräsentiert werden oder wo typische Eingabegeräte wie Maus und Tastatur schlecht oder gar nicht verwendbar sind, wie etwa im Umfeld von VR/AR, ist es wichtig, effizientere und natürlichere Benutzerschnittstellen zu entwickeln und bereitzustellen.

Da multimodale Dialogsysteme auf eine möglichst natürliche Konversation zwischen virtuellen und echten Menschen abzielen, ist hier der Dialogbegriff wie von der Alltagssprache her gewohnt aufzufassen. Multimodale Dialogsysteme erweitern damit Sprachdialogsysteme (wie man sie etwa von Support-Hotlines kennt) um weitere Modalitäten, welche sich auf die fünf menschlichen Sinne beziehen, d.h. das Dialogsystem weist nun neben der auditiven mindestens auch eine graphische Komponente auf. Das Dialogmanagement muss damit nicht mehr nur die Sprache selbst, sondern daneben auch die zugehörige Körpersprache erzeugen (vgl. Abb. 1.1), was einen Paradigmenwechsel von der Generierung natürlicher Sprache hin zur Generierung von multimodalem Verhalten bedeutet [326].

Multimodale Dialogsysteme, die mit Hilfe virtueller Charakteren modelliert werden, stellen jedoch nach wie vor ein Problem dar, das trotz einer Vielzahl von Forschungsarbeiten auf diesem Gebiet aufgrund der Komplexität der Thematik bis heute nicht vollständig gelöst ist. Während sich die Künstliche Intelligenz (KI) hierbei primär auf Persönlichkeitsmodellierung, Dialoggenerierung sowie Sprachverstehen und Sprachsynthese konzentriert, beschäftigt sich die Computergraphik mit der visuellen Repräsentation der Charaktere. Dazu gehören neben Animationsaspekten auch die Bereiche Rendering und Simulation, wobei die verschiedenen Gebiete jedoch meist für sich alleine betrachtet werden.

Dennoch stellen virtuelle Charaktere ein geeignetes Ausgabemedium für multimodale Systeme dar, da sie eine symmetrische Multimodalität ermöglichen [326], weil alle Eingabemodalitäten auch für die Ausgabe verfügbar sind. Durch kommunikativ effektiv und der Situation angemessen reagierende persönliche Dialogpartner lässt sich somit eine neue Qualität von interaktiven Systemen erreichen. Glaubwürdigkeit und Erfolg solcher Benutzerschnittstellen hängen dabei jedoch von vielen Faktoren ab [321]. Neben der Gesprächskompetenz der Konversationsagenten und deren Fähigkeit, innerhalb des gegebenen Kontextes plausibel zu handeln, zählen desweiteren die sinnvolle Reaktion auf Benutzeraktionen und die Umgebung dazu, ein glaubwürdiges und konsistentes Bewegungsverhalten sowie visuelle Akzeptanz, und nicht zuletzt Effizienz bei der Anwendungserstellung.

Das Verhalten kann dabei wie beim Game Design entweder direkt von Gestaltern geplant oder autonom von KI-Systemen generiert werden. Auch wenn der Einbezug einer glaubwürdigen Verhaltenssimulation den Rahmen dieser Arbeit sprengen würde, so sind entsprechende Parameter und mögliche Randbedingungen bei der nachfolgenden Darstellung doch zu berücksichtigen. Von daher ist gerade in diesem Umfeld die Vollständigkeit des zugrundeliegenden Rendering- und Animationssystems ein wesentlicher Gesichtspunkt.

So sieht man zunächst durch die Unterhaltung mit dem Konversationsagenten, anders als z.B. bei einem typischen Computerspiel, den als Userinterface dienenden Avatar aus größerer Nähe. Auch wenn sich die Forschung in dieser Hinsicht bislang meist nur auf Gestik, Mimik und Sprache konzentriert hat, so ist es doch erforderlich, im Sinne eines Komplettsystems den gesamten virtuellen Menschen zu betrachten. Dazu zählen auch bisher vernachlässigte Aspekte wie dynamische Hautveränderungen aufgrund (psycho-) physiologischer Prozesse oder auch Kameraführung und die gesamte virtuelle Umgebung, wobei die beiden letzten Punkte als weiteres Kommunikationsmedium dienen können.

Auf Seiten der Visualisierung ist es damit wichtig, ein breites Spektrum von in Echtzeit frei miteinander kombinierbaren Aktionen und Fähigkeiten bereitzustellen und koordinieren zu können. Zudem ist heutzutage aufgrund immer leistungsstärkerer Graphikkarten gerade beim Rendering ein erstaunlicher Grad an Realismus erreichbar. Dies erlaubt ein neuartiges Characterdesign, bei welchem z.B. auch physiologische Prozesse mit einbezogen werden können, was letztlich auch eine neue Form von Interfaces ermöglicht. Dabei sind für dialogartige Systeme neben inhaltlichen Aspekten die Glaubwürdigkeit und Konsistenz des Verhaltens wesentlich, da man als Mensch unbewusst darauf genauso wie auf den zunächst offensichtlich scheinenden Inhalt achtet. Daneben sind aber auch genauso externe Faktoren wie Beleuchtung oder Kameraeinstellung wichtig für die Perzeption.

Fokus der Arbeit

Das Ziel dieser Arbeit ist somit die Konzeption und Entwicklung eines generischen Systems für die Präsentationskomponente multimodaler Dialogsysteme, welches relevante Funktionalitäten mit aufeinander abgestimmten Verfahren integriert und in entsprechender Form handhabbar bereitstellt. Das System soll sich dabei in etablierte Visualisierungstechniken, bestehende Standards und nicht zuletzt in auf abstrakterer Ebene arbeitende Modularchitekturen durch die Bereitstellung einer darauf aufsetzenden Interfacesprache integrieren lassen. Auf der Szenengraphebene werden dazu alle relevanten Basisfunktionalitäten als

Abbildung 0.2: *Wichtige Bausteine für glaubwürdige, interaktive Charaktere: Haarsimulation, Weinen, dynamisch erzeugte Gesten, deklarative Kamerakontrolle (hier mit der Vorgabe, von der geg. Kameraposition aus den Kopf in der Mitte zu halten).*

Bausteine zur Dialogsystemerstellung bereitgestellt, welche dann entweder direkt zur Entwicklung vorgeskripteter Dialoganwendungen verwendet werden können, oder welche über eine deklarative Kontrollschicht von KI-Modulen genutzt werden können.

Wirklich interaktive virtuelle Charaktere erfordern damit ein hohes Maß an Kontrolle über deren Pose, Gestik, Mimik und Sprache sowie generell über deren physiologischen bzw. nach außen hin wahrnehmbaren Prozesse (siehe Abbildung 0.2). Neben elementaren Visualisierungs- und Animationsfragestellungen müssen daher auch Abläufe koordiniert werden. Um nun weiterhin die Funktionalität eines Szenengraphen bzw. ganz allgemein eines Visualisierungssystems auch anderen Benutzergruppen außerhalb der Computergraphik zur Verfügung zu stellen, muss dessen Struktur handhabbar gemacht werden.

Ein wesentliches Problem ist allerdings die gegenseitige Interdependenz der einzelnen Aspekte, welcher man heute zumeist mit mehrschichtigen Architekturen zu begegnen versucht. Wenn es seitens der Computergraphik gelingt, ein standardisierbares System zu entwickeln, das einerseits möglichst vollständig hinsichtlich der Fähigkeiten und Fertigkeiten eines solchen Avatars ist, und das andererseits eben diese Funktionalitäten über abstraktere Schnittstellen auf zugängliche Weise anderen Komponenten bereitzustellen vermag, wäre dies eine wesentliche Grundlage zur Entwicklung von interaktiveren und gleichzeitig natürlicher erscheinenden virtuellen Agenten und würde mit dazu beitragen, die hier noch immer existierende Kluft zwischen KI und Graphik zu überbrücken.

Rahmensystem und Kontrollschicht

Auch wenn die oben angesprochenen Visualisierungstechniken beispielsweise für KI-Systeme o.ä. interessant sind, so ist doch der Szenengraph dafür ungeeignet. Deshalb ist es hinsichtlich einer modularen Architektur notwendig, eine geeignete Abstraktion zu finden, die mit Szenenbeschreibungen und Animationen sowie deren Synchronisation und Kombination umzugehen vermag (vgl. Abbildung 1.4 auf S. 28). Hierzu bieten sich etwa High-Level Sprachen zur Steuerung der Charaktere an, wobei in diesem Zusammenhang allgemein die Koordination und Synchronisierung von Abläufen von Interesse ist.

Deshalb ist es zunächst wichtig, geeignete Interface- und Kontrollsprachen zu definieren, welche KI- und anderen Systemen, die nicht auf Polygonebene sondern wie die KI

mit Dialogelementen für intelligente Agentensysteme arbeiten, einen Zugang erlaubt und somit auf höherer Ebene eine Definition und Steuerung der Verhaltensbeschreibung ermöglicht. Dabei werden Animationen u.ä. als Service benutzt, weswegen für einen Dialog der Ablauf von Ereignissen und komplexen Animationen gesteuert und koordiniert werden muss. Dazu dient die in dieser Arbeit vorgestellte Interface- und Kontrollsprache PML als zusätzliche Schicht um die Präsentationskomponente, um deren Dienste anderen Anwendungen zur Verfügung zu stellen. Anforderungen wie etwa das Mischen und Überblenden von Animation lassen sich damit aus höheren Schichten ableiten.

Ein virtueller Charakter ist als User-Interface-Komponente in der Regel in eine größere Anwendung bzw. Umgebung eingebettet. Diese Einbettung erfordert die Berücksichtigung von verschiedenen Integrations- und Standardisierungsaspekten. Auf unterster Ebene bedeutet dies die Integration neuer Simulations- und Visualisierungsverfahren in etablierte Visualisierungstechniken wie den Szenengraph. Gerade im Characterbereich und vermehrt mit der Entwicklung moderner, GPU-basierter Verfahren findet man immer häufiger eine gewisse Dualität von Simulation und Rendering. Um derartig aufeinander abgestimmte und zusammengehörige Simulations- und Renderingtechniken nutzen zu können (etwa im Falle einer Haar- oder Tränensimulation), ist die traditionelle Traversierung des Szenengraphs hier aufzubrechen und für den Anwendungsentwickler transparent zu erweitern.

Für einen effizienteren Datenaustausch und um anderen Komponenten eine einfache Möglichkeit zu geben, auf der Präsentationskomponente aufzusetzen, sowie für eine größere Nachhaltigkeit der entwickelten Verfahren, ist die Einbettung in bestehende Standards ein zentraler Gesichtspunkt. Nutzt man echte ISO-Standards wie X3D [336], welches sowohl als deklarative Szenengraph-basierte Applikationsbeschreibungssprache, als auch als Laufzeitumgebung für die Entwicklung interaktiver 3D-Anwendungen zu verstehen ist, so ist gerade hinsichtlich einer multidisziplinären Zusammenarbeit eine einfachere Systemintegration möglich. Da die Humanoid Animation (H-Anim) Komponente [335] als Teil von X3D die Thematik technologisch aufgreift, ist es in diesem Zusammenhang sinnvoll, hier anzusetzen und das Applikationsmodell an dieser Stelle zu erweitern.

Die hier normalerweise verwendeten Standard-Animations-Engines weisen allerdings meist keine High-Level Schnittstellen auf. Aufgrund des in diesem Bereich vorzuziehenden lose gekoppelten, serviceorientierten Ansatzes sind Interfacesprachen zur Modulkommunikation jedoch besser geeignet als API-Aufrufe. Von daher wird in dieser Arbeit zunächst ein kurzer Überblick über entsprechende Ansätze aus dem Gebiet der Markup-Sprachen für High-Level-Kontrolle und Repräsentation virtueller Charaktere gegeben. Hier herrscht eine große Mannigfaltigkeit vor. Dies reicht von Sprachen, die auf sehr abstrakter Ebene lediglich Ziele vorgeben, bis hin zu solchen, die es erlauben, die einzelnen Freiheitsgrade des Bewegungsmodells direkt zu setzen. Bekannte Beispiele sind die auf XML beruhende Virtual Human Markup Language (VHML), sowie die etwas neuere Interfacesprache Behavior Markup Language (BML) [318], bei der u.a. versucht wird, bisherige Ansätze zu vereinen. Typischerweise sind diese Sprachen sehr domänenspezifisch und dienen primär zur Modellierung multimodaler Dialoge auf einer recht abstrakten Ebene.

Von daher wurde kooperativ die ebenfalls auf XML basierende Auszeichnungssprache PML (Player Markup Language) entwickelt. Hierbei gibt die Präsentationskomponente ihre Funktionalität im Sinne eines zusätzlichen Dienstes über PML nach außen an externe

Module weiter. PML erlaubt es damit, Charakteranimationen zu steuern und zu synchronisieren, sowie verschiedene Animationen bzw. Aktionen miteinander zu kombinieren. Im Sinne einer serviceorientierten Architektur ist die Sprache zunächst von der Implementierung des verwendeten Rendering-Systems unabhängig. Dabei erfolgt das Mapping von High-Level Anforderungen auf die konkreten Szenengraph-Elemente unter Zuhilfenahme eines sogenannten Gesticons (vgl. [194]). Ein PML-Skript bezieht sich dabei immer auf einen bestimmten virtuellen Charakter oder ein anderes 3D-Objekt.

Es wird primär zwischen Definitions- und Actions-Skripten unterscheiden. Nachdem die Definitions-Skripte geladen wurden, welche die vorhandenen Elemente einer Szene beschreiben, lassen sich anschließend in den Actions-Skripten die Animationen sowie andere Eigenschaften der 3D-Objekte (z.B. Position oder Sichtbarkeit) referenzieren. Mithilfe eines Actions-Skripts können die Animationen der virtuellen Charaktere in eine zeitliche Reihenfolge gebracht und deren Bewegungen synchronisiert werden. Animationen und andere Ereignisse können parallel oder sequenziell angeordnet werden und zudem kann die Dauer der jeweiligen Animation festgelegt werden. Die Verarbeitung der PML-Skripte wird dabei über Nachrichten gesteuert, d.h. über Message-Skripte.

Bausteine und Ausführungsschicht

Zur Umsetzung der über die deklarative Kontrollschicht gegebenen Anweisungen wurde im Rahmen dieser Arbeit desweiteren ein um neue Animations- und Simulationstechniken erweiterbares Rahmenwerk auf Basis des X3D-Standards konzipiert und in das Instant Reality Framework [135] integriert, welches X3D als Applikationsbeschreibungssprache nutzt. Dabei wurden auch entsprechende neuartige Echtzeit-Renderingverfahren entwickelt, welche insbesondere die Darstellung psycho-physiologischer Phänomene wie Erröten, Blasswerden, Schwitzen oder Weinen umfassen, und welche wesentlich sind für eine glaubwürdige, emotional überzeugende und damit konsistentere Darstellung des Avatars.

Daneben wurden auch externe Faktoren wie Beleuchtungsaspekte, geeignetere Möglichkeiten zur Kamerasteuerung sowie die ganze (virtuelle oder im Falle von AR reale) Umgebung berücksichtigt. Damit stellt die hier entwickelte Präsentationskomponente nicht nur vom Menschen i.A. bewusst steuerbares Verhalten wie Pose, Gestik, Mimik, Augenbewegungen und Sprache als notwendige Voraussetzung zur Visualisierung bereit, sondern erlaubt darüber hinaus die Nutzung von unbewusst ablaufenden bzw. vom Menschen nicht kontrollierbaren Prozessen wie Weinen und Erröten aber auch von Haarbewegungen usw. Abbildung 1.9 auf S. 36 zeigt dabei die Einordung in aktuelle Forschungsfelder.

Character Dynamics und Haarsimulation

Wie eingangs erwähnt, beschreibt die H-Anim-Komponente, als Teil des X3D-Standards, einen portablen Standardavatar mit wohldefinierter Skelett-Struktur, wobei die Hierarchie der Knochen und Gelenke die Szenengraphenstruktur erweitert und auf einer sog. “Skin and Bones” Beschreibung beruht. Die Animation geschieht dabei über X3D-Interpolator-Knoten. Für einfache Animationen reicht diese Methode aus, aber wenn mehrere Animationen gleichzeitig abgespielt werden sollen, dann mangelt es an Mechanismen zum Mischen und Überblenden. Zudem fehlt in X3D nicht nur eine darauf aufsetzende API

zur Animationskontrolle, und somit ein einheitliches Konzept zur Verhaltensmodellierung und zum Authoring, sondern beispielsweise auch so grundlegende Bausteine wie Text-To-Speech (TTS) und damit verbunden eine passende Lippenbewegung für Sprache.

Zur Umsetzung der Animationskontrolle wurde daher zunächst ein Animationssystem konzipiert, welches über PML gesteuert wird und den X3D-Standard erweitert. Die Kommunikation mit dem Animationssystem erfolgt dabei über den neuen *TimelineComposer*-Knoten, der intern PML-Commands an den PML-Parser weitergibt und PML-Messages entgegennimmt. Die zeitliche Ansteuerung der Animationen und anderer Ereignisse erfolgt dabei über den Scheduler, der für die Umsetzung aller Aktionen und der damit verbundenen Definitionen sorgt. Bewegungen des Körpers erfolgen in der Regel über Keyframe-Animationen, während das Gesicht mit Hilfe von Morph-Targets animiert wird.

Anhand von Namen und aktueller Zeit holt sich der *AnimationController* dazu die Animationsinformationen aus den *AnimationContainer*-Knoten, mischt und überblendet sie gegebenenfalls und schreibt das Ergebnis dann in die Gelenkknoten. Dabei wird immer dann gemischt, wenn sich verschiedene Bewegungen zeitlich überlappen, während zwischen aufeinanderfolgenden Animationen dynamisch überblendet wird, indem die jeweiligen Gelenkstellungen durch eine Ersatzbewegung einander angenähert werden. Auf diese Weise wird eine nach außen hin komplexere Architektur vermieden und die Behandlung verschiedener Animationstypen vereinheitlicht. Somit kann z.B. auch durch Mischen zweier verschiedener Zeigegesten leicht eine dazwischenliegende Stellung erreicht werden.

Ein flexibles Animationssystem erfordert darüber hinaus die Simulation resultierender Abhängigkeiten wie etwa Haarbewegungen, z.B. wenn ein virtueller Mensch seinen Kopf bewegt oder sich nervös mit der Hand durch sein Haar fährt, was damit ein wesentlicher Bestandteil einer Geste sein kann. Zumindest längeres Haar muss sich also auch bewegen können, um natürlich zu erscheinen. In der Literatur finden sich verschiedene Ansätze zur Lösung dieses Problems, angefangen von Keyframe-Animationen über Masse-Feder-Systeme bis hin zu Vektorfeldern. Während Animationen zwar schnell sind, aber keine anderen als die vorberechneten Bewegungen erlauben, sind die meisten anderen Ansätze nicht mehr echtzeitfähig. Von daher wurde ein System zur dynamischen Echtzeitsimulation von Haaren auf Basis einer offenen seriellen kinematischen Mehrkörperkette entwickelt.

Das kinematische Verfahren ist dabei nicht nur stabiler als ein zu Vergleichszwecken entwickeltes Verfahren auf Basis eines gedämpften Masse-Feder-Systems, sondern reagiert in seinen Bewegungen auch deutlich naturgetreuer. Die vereinfachte Kollisionsbehandlung erfolgt auf Grundlage parametrischer Kollisionsobjekte, was bei geschickter Platzierung nicht auffällt, aber die Rechnung deutlich beschleunigt. In Verbindung mit einem speziell entwickelten Haarshader lassen sich somit ansprechende Frisuren in Echtzeit darstellen.

Prinzipiell werden bei der Simulation zwei verschiedene Objekttypen unterschieden; zum einen die so genannten Ankerpunkte, die als Wurzel der kinematischen Kette dienen, und weiterhin die normalen Massepunkte, die als Gelenkknoten fungieren. Das Rendering erfolgt dabei mit Hilfe von Quadstrips. Um nun von einer linienartigen Kettenstruktur auf eben diese Polygonstruktur zu kommen, besitzt jeder Punkt der Kette einen zugeordneten Punkt, auf den die während der Simulation erfolgte Positionsänderung durch Addition eines zuvor am Kopf ausgerichteten Differenzvektors übertragen wird. Die Simulation selbst, deren wichtigste Eigenschaft der Erhalt der ursprünglichen Länge zwischen den

einzelnen Massepunkten ist, erfolgt schließlich durch Iteration des Gesamtsystems.

Darstellung emotional verursachter Hautveränderungen

Wesentliche Aspekte zwischenmenschlichen Verhaltens sind auch das Zeigen und Interpretieren von Gefühlen. Gesicht und Mimik geben dabei Auskunft über die individuelle Gefühlslage und Eigenheiten eines Menschen und sind damit ebenfalls sehr wichtig für die Kommunikation. Daher wirken Gesichtsanimationen durch die Entwicklung besserer Animationstechniken, meist über Motion Capturing mit geeigneter Nachbearbeitung, inzwischen recht wirklichkeitsgetreu, da hier zudem auf psychologisch fundierte Modelle wie z.B. das sog. Facial Action Coding System (FACS) [77] aufgesetzt werden kann.

Da man es gewohnt ist, gleichermaßen auf solche nonverbalen Signale zu achten (vgl. Abbildung 1.1), können animierte Charaktere durchaus emotionale Reaktionen beim Benutzer auslösen. Je menschenähnlicher virtuelle Charaktere werden, desto höher sind allerdings die Erwartungen an Natürlichkeit sowie Synchronizität zwischen verschiedenen Modalitäten (z.B. Sprache und Mimik), und desto eher verursacht ein vom menschlichen Normalverhalten noch abweichendes Ausdrucksverhalten Abneigungen beim Betrachter – ein Effekt der auch als "Uncanny Valley" bezeichnet wird [221].

Daher muss die Darstellung emotionaler Ausdrücke synchron zu Sprache und Körperanimationen kontrollierbar sein. Die meisten Animationsmodelle beschränken sich allerdings nur auf räumliche Veränderungen. Dekodiermodelle wie FACS zur Kennzeichnung emotionaler Gesichtsausdrücke beziehen sich zudem nur auf eindeutig zuordenbare Bewegungen im Gesicht und lassen vaskuläre u.ä. Veränderungen unberücksichtigt. Eine Herausforderung ist es daher, interaktiven Charakteren realistischer wirkende Gefühle zu verleihen, etwa durch Erröten oder Weinen zur Darstellung starker Emotionen. Solche physiologischen Reaktionen wurden bisher jedoch meist vernachlässigt oder künstlerisch angegangen.

Gerade diese dynamischen Veränderungen der Haut selbst lassen aber starke Emotionen bei virtuellen Charakteren lebendiger und glaubwürdiger wirken und sollten daher auch Berücksichtigung finden. Im Gegensatz zu Gesichtsanimationen über Morph-Targets usw. gibt es hierfür bisher jedoch noch nichts vergleichbares zu FACS, d.h. zur Simulation solcher psycho-physiologischer Effekte existieren zum einen kaum Ansätze und zum anderen auch keine Modelle, obwohl dies mit dazu beitragen kann, simuliertes Verhalten besser wahrzunehmen. Damit einher geht also die Realisierung eines Modells zur Klassifizierung und Parametrisierung dynamischer, emotional verursachter Hautveränderungen.

Das entwickelte Klassifizierungsmodell beruht auf den Emotionsmodellen von Ekman [77] und Plutchik [249]. Im Gegensatz zu Plutchiks Modell ist das ausgearbeitete Modell aber lediglich zweidimensional und hat seinen Fokus auf den Hautveränderungen, die bei bestimmten Emotionen im Gesicht auftreten, und gehört damit zu den Dekodiermodellen. Die emotionalen Farbänderungen lassen sich dabei einfach in bereits vorhandene Hautrendering-Verfahren integrieren. Hierdurch können starke Emotionen, die mit Erröten, Erblassen oder Weinen einhergehen, auf einem 3D-Menschmodell einfach und glaubwürdig umgesetzt werden und bilden mit der Mimik eine in sich konsistente Animation.

Um festzustellen, wie glaubhaft die erzielten Ergebnisse sind, wurde eine Evaluation mit 57 Personen im Alter von 7 bis 65 Jahren durchgeführt. Hierbei wurde deutlich, dass Gesichtsausdrücke mit farblichen und ähnlichen emotional verursachten Veränderungen

Abbildung 0.3: *Anders als in Abb. 0.1 gezeigt, sollte insbesondere bei Mixed Reality Anwendungen die Darstellung der virtuellen Charaktere auch zur Umgebung passen.*

besser erkannt werden als solche, die nur reine Geometrieanimationen beinhalten.

Die Animation von Tränen u.ä. geschieht mit Hilfe einer rein im Bildraum erfolgenden Tröpfchenfluss-Simulation. Auch dazu wird ein Konzept und dessen Umsetzung zur grafikkartenbasierten Echtzeit-Simulation von tröpfchenförmigen Flüssigkeiten dargelegt. Das vorgestellte Modell bildet eine hier Surface-Map genannte Textur auf die Oberfläche eines 3D-Modells ab. Jeder Texel dieser Textur repräsentiert ein Fluidelement und für jedes Fluidelement wird auf der GPU dessen Fließgeschwindigkeit und eine auf diesem wirkende externe Kraft simuliert. Dabei können Tropfen, wobei auch deren Randwinkel berücksichtigt werden, auf annähernd beliebigen Modellen animiert werden.

Kamera und Beleuchtung

Szene und Kameraeinstellung sind orthogonal zueinander zu sehen. Gerade Effekte wie leichtes Erröten oder Weinen sind z.T. nur bei extremen Nahaufnahmen zu erkennen, aber generell kann die Wahl der Kameraeinstellung die Perzeption deutlich beeinflussen [44]. So können subtile Unterschiede in z.B. Kameraposition und -winkel bereits durchaus die Bedeutung einer Szene verändern. Dementsprechend wurde eine virtuelle Kamera entworfen und implementiert, welche statt mit der üblichen Kamerapose entsprechend bewährter kinematographischer Prinzipien parametriert und animiert werden kann.

Daneben ist auch die Wahl von Linse, Blende, Filter usw. wesentlich. Ein solches zusätzliches Element ist z.B. die Tiefenschärfe. Diese kann nicht nur für eine realistischere Darstellung verwendet werden, sondern auch um gezielt die Aufmerksamkeit des Zuschauers auf wichtige Elemente zu lenken bzw. unwichtige Elemente durch gezielte Unschärfe in den Hintergrund zu rücken. Mit anderen Bildraumfiltern lassen sich dabei in ähnlicher Weise verschiedene Linsentypen nachahmen, aber auch Sepia-Effekte usw. Diese visuellen Effekte sind dabei als Teil der zuvor genannten kinematographischen Kamera realisiert.

Um dem gerade aktuellen AR-Trend zu begegnen, wurden weiterhin Verfahren zur Beleuchtungs- und Materialrekonstruktion im Kontext von Mixed Reality vorgestellt und untersucht. Die korrekte Beleuchtung virtueller 3D-Objekte durch eine Ableitung der Lichtverhältnisse aus der Realität fällt der Beleuchtungsrekonstruktion zu. Weil dies bei typischen AR-Anwendungen meist unberücksichtigt bleibt, sind virtuelle Objekte durch ihr unechtes Aussehen deutlich von der Umgebung zu unterscheiden. Die Beleuchtungsrekonstruktion ist daher ein extrem wichtiger Bestandteil der Simulation, wenn ein echtes Mixed Reality Gefühl entstehen soll (vgl. Abbildung 0.3), wobei ein physikalisch-basierter

Global Illumination Ansatz in der Praxis jedoch nicht in Echtzeit anwendbar ist.

Stattdessen wird in dieser Arbeit gezeigt, dass Image Based Lighting (über sog. Irradiance-Maps), verbunden mit schnellen Schattierungs- und Schattenwurf-Verfahren, wie Ambient Occlusion und Softshadows, eine sinnvolle Kombination darstellt und gegenüber aufwendigeren Verfahren wie etwa PRT in Bezug auf Dynamik überlegen ist. Verzichtet man auf Spezialeffekte, so wird mit der geschickten Kombination dieser Verfahren ein guter Mittelweg eingeschlagen, der Performanz bei gleichzeitig hoher Darstellungsqualität liefert und auch auf durchschnittlicher Graphikhardware durchführbar ist.

Fazit

Im Rahmen dieser Arbeit wird nun ein generisches System für die Präsentationskomponente multimodaler Dialogsysteme vorgestellt, welches wesentliche Funktionalitäten integriert und diese handhabbar bereitstellt. Es verbindet dabei wichtige Bausteine in der Szenengraph-basierten Ausführungsschicht mit einer deklarativen Kontrollschicht und ermöglicht somit glaubhaft reagierende Charaktere in dynamischen Umgebungen.

Hierfür werden damit zusammenhängende Teilfragestellungen ausgearbeitet, insbesondere die Themenfelder Animations- und Kamerakontrolle, Haarsimulation, Emotionsvisualisierung (unter besonderer Berücksichtigung psycho-physiologischer Phänomene) sowie Mixed Reality Aspekte. In diesem Zusammenhang wird ein Verfahren für konsistentes Rendering mit der restlichen Szene bei AR-Anwendungen vorgestellt. Die beschriebenen Forschungsergebnisse beruhen dabei größtenteils auf den ab Seite 256 aufgelisteten Publikationen.

Durch die Integration der entwickelten Konzepte und Verfahren in das Instant Reality Framework [135] wird zudem eine größere Anwendungsbandbreite ermöglicht. Verfügbarkeit und Effizienz werden weiterhin durch die Integration in den X3D-Standard erreicht. Dabei sind die entwickelten Bausteine skalierbar und vielseitig genug, um auch in einem anderen Kontext nutzbar zu sein. Durch den Einbezug der Darstellung psycho-physiologischer bzw. ganz allgemein “unbewusst” ablaufender Prozesse bieten sich darüber hinaus neue Evaluationsmöglichkeiten durch ausdrucksstarke virtuelle Charaktere. Schließlich wird der Nutzen des Systems in verschiedenen Anwendungsfeldern exemplarisch gezeigt.

Contents

1 Introduction

During the past few years there has been an increasing interest in virtual characters [109], not only in Virtual Reality, computer games or online communities such as Second Life, but also for dialog-based systems like tutoring systems or edutainment and infotainment applications. This is directly associated with the major challenges of Human-Computer-Interface (HCI) technologies in general and immersive Mixed and Virtual Reality (VR) concepts in particular, as they are both aimed at developing intuitive man-machine-interfaces instead of the standard WIMP[1] style of human-computer interaction, which basically has not changed for more than three decades.

However, since computing power becomes more and more ubiquitous, it is inevitable to extend these traditional interaction methods. In this regard, for many application areas virtual characters are well suited as an intuitive human-computer-interface by simulating verbal as well as nonverbal communicative behavior (as shown in Figure 1.1), for ensuring intuitive interactions even for inexperienced users and beyond standard settings. Possible fields of application embrace such situations, where a task is most naturally represented by talking or where typical input devices like mouse and keyboard are difficult to use or not available at all, such as in immersive VR settings or in Augmented-Reality-supported on-site manuals for maintenance scenarios.

Other examples are assistance systems such as virtual tour guides or interactive manuals, where the virtual human explains cultural heritage sites or the usage of a new device. Somewhat unsuitable areas for conversational interfaces are for instance typical office

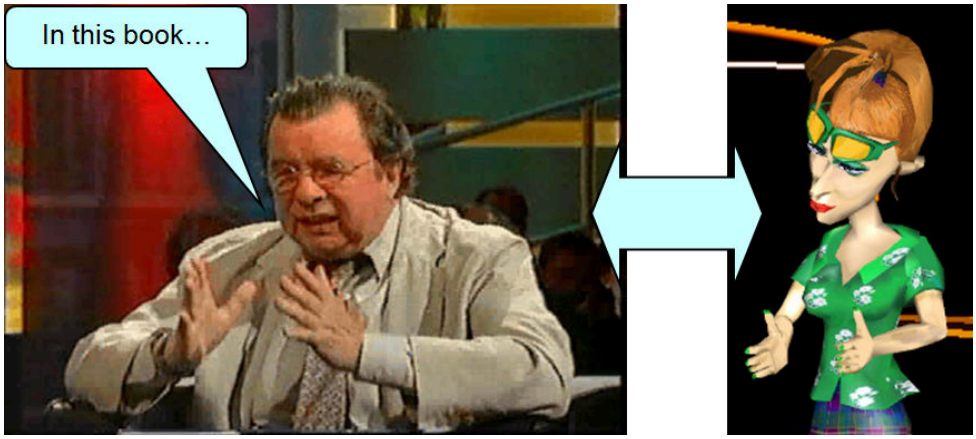

Figure 1.1: *Example of nonverbal communication: on the left a screenshot of a TV talk show is shown and on the right a virtual character that is gesticulating in a similar way.*

[1] Window, Icon, Menu, Pointing device; i.e. standard GUI (Graphical User Interface) interaction

applications like wordprocessing and applications that are mainly driven by direct manipulation tasks such as painting or moving around photos or architectural blueprints on a multi-touch table. However, such gestural input is just another mode of communication and thereby part of a multimodal interface system.

1.1 Motivation

Multimodal user interfaces as well as Augmented and Virtual Reality technologies in general are an object of active research and development. But due to the complexity of these topics and the variety of possible input and output devices, integrated VR/AR systems are required to ease application development and interface design. Hence, for instance most of the results that were achieved in this area by the department "Virtual and Augmented Reality" at Fraunhofer IGD are integrated into the Mixed Reality system Instant Reality [135, 22]. These research results include technologies in the areas of multi-display VR environments [24, 23], scalable high-quality rendering, multimodal interaction techniques like haptics or multi-touch [152, 153], and Computer-Vision-based tracking (e.g. [27, 163]) for interactive Augmented Reality (AR) applications.

Here, the term Mixed Reality (MR) needs some further explanations. As is visualized in Figure 1.2, MR spans the whole continuum between Virtual Reality (VR) and Augmented Reality [137]. The left image shows an immersive VR design review application deployed in IGD's CAVE, and the right image shows an example of an outdoor system for AR-enhanced sightseeing, where ruins are augmented with additional 2D and 3D information. Whereas VR only deals with solely virtual objects and data, AR aims at integrating additional data such as text or images and virtual 3D objects into real scenes (e.g. for assembly simulations). In this respect, this entire field of analyzing and synthesizing image data, including the way how one can interact with it, is called Visual Computing.

Further, especially in mobile computing in combination with geolocation-based services, there is a recent trend in augmenting the real world with virtual information, which is made possible due to increasing processing power, bandwidth, and 3D capabilities even on mobile devices. Thereby, fascinating new user experiences come within reach, where e.g. a virtual character, as an augmented master teacher, mediates procedural knowledge like how to use a newly bought real device. This at first requires that a live video of the real scene needs to be put behind virtual objects and thereby the exact pose of the user or camera somehow needs to be determined with vision-based tracking techniques, which is out-of-scope here and assumes to be given within the course of this thesis.

However, if we want to augment the images not only with some additional information but also with nice looking 3D objects and virtual agents, the question arises how to fit them as seamless as possible into real scenes [151]. Therefore, besides the geometric registration, realistic and photometrically consistent lighting is also needed, and the changes in the real lighting conditions caused by the virtual objects need to be updated as well. So, correct shading gets even more of an issue, if Mixed instead of pure Virtual Reality scenarios are to be considered. This also includes occlusion handling as well as the reconstruction and simulation of real-world lighting situations for correct shadows and so on [96].

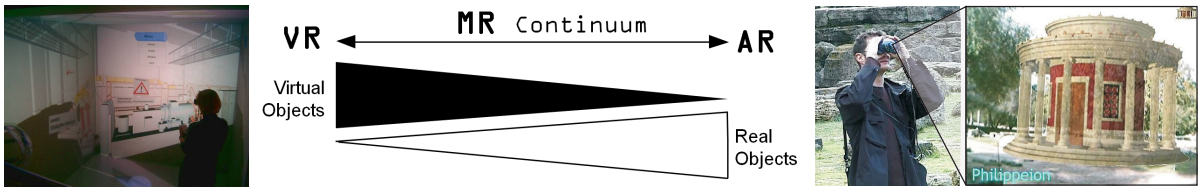

Figure 1.2: *Mixed Reality (MR) spans the whole continuum between VR and AR technologies.*

Another important aspect in virtual environments in general are virtual humans. Only those virtual worlds are able to increase the sense of "presence" within immersive virtual environments, which contain "living" things and are inhabited by virtual characters, that act as believable and autonomous as possible [109]. This is usually achieved by separating mind and body, which is nicely visualized with the marionette in the left part of Figure 1.7. Modeling the character's "mind" embraces the simulation of conscious and emotional behavior but is beyond the scope of this thesis, whereas on the "lower" graphics level believability means a plausible appearance and animation of the body.

1.1.1 Multimodal Dialog Systems

Albeit the architectural design of full-blown dialog systems as well as accompanying topics like natural language processing, discourse planning and the like is beyond the scope of this thesis and in the following assumed to be given, a brief overview and definition of relevant terms is necessary to understand the research field. Instead, we will focus on the renderer component that is responsible for the visual and auditive output. Though GUI forms sometimes are called a dialog (yet in an abstract sense they are), in the context of dialog systems the term dialog is meant with a much broader scope in that the system is designed for the conversation with a human being in a natural manner concerning the input and output channels [343]. This includes text, like in older role-playing games, but also speech, such as automated technical support per telephone.

There exist different types of systems, which can be classified according to device (e.g. in-car systems), modality (e.g. text-based vs. multimodal), or application area (e.g. edutainment). The key component of a dialog system is the dialog manager [36, 343], which plans the abstract state and strategy of a dialog with Artificial Intelligence techniques. An example is the J-Alice [263] toolkit for so-called chat bots that leverages concepts of natural language processing, as first presented with the Eliza program [342]. Today (chat) bots are omnipresent in the Internet[2] like in online-communities or for advertisements and online help. In this context, these bots can be seen as conversational agents – or embodied conversational agents (ECA) in case they have a 2D or 3D visual representation.

In the latter case, the dialog system consists at least of an auditive and graphical component, and often include other kinds of modalities such as touch-based input interfaces or gestural interaction. Multimodal dialog systems thereby extend typical speech dialog systems with additional modalities just like human-human interaction [36, ch. 5], whereas

[2] A funny example of a faked Captain Kirk chat bot as talking head can be found here: http://showcase.pandorabots.com/pandora/talk?botid=d10d53a63e345abf&skin=iframe

the term modality refers to the five human senses [326]. Similar to face-to-face dialogs, communication is based on speech and nonverbal communication like facial expressions, gaze, gestures and postures, which requires that the dialog management not only generates speech but also the corresponding body language, which implies a shift from natural language generation to multimodal behavior generation. In this regard we use the term "multimodal dialog system" as defined by Wahlster [326].

Here, ECAs are autonomous entities with communicative and emotional capabilities that can serve as pedagogical agents, web assistants, etc., whereas research generally focuses on how to specify and control their behavior. By simulating communicative behavior including verbal and nonverbal communication, which is visualized in Figure 1.1, natural user interfaces can thus be provided (Figure 1.3 shows an example application). As can be seen, not only the dialog behavior between other characters and real humans but also rendering and animation aspects are important for believability and consistency.

Suitable interaction metaphors for dialog-based systems are "guidance", a concept that can be achieved by narration and thus digital storytelling techniques but is still a bit controversial [204], and "natural dialog" (by providing conversational user interfaces with responsive virtual humans) – an ability that people practice every day and in every face-to-face conversation. Both previously mentioned concepts are typical areas of research in Interaction Design on the one hand and Artificial Intelligence (AI) on the other hand, and they usually follow a goal- and communication-driven approach [104]. This is often achieved in combination with an ontology for knowledge management (which formally represents the concepts of a certain domain including their relationships) by first defining certain goals and dialog acts on a very high level of abstraction (e.g. "Explain usage of device X"), which are then further refined and enhanced with concrete behavior [204].

The refinement is mostly done by different loosely coupled modules that are responsible for dialog generation, speech synthetization, gesture control, adaption to various emotions, etc. [128]. In chapter 7.3.3 on page 216 such a module pipeline (which was developed within the Virtual Human project [321]) is exemplarily described. Typically, each module adds more information until the result is concrete enough for being visualized by a rendering engine like in the example application shown in Figure 1.3.

Scope of this Research

As said, this work focuses on the visualization component of multimodal dialog systems, which in the latter context is also known as the surface realizer [204, 123] and mainly deals with the graphical realization of the embodied agent and its nonverbal output.

In principle, this defines the requirements for such a component but also the scope of the research. For instance, in dialog-like applications such as in infotainment only one or a few characters are used. Therefore, e.g. crowd rendering [309] and the like does not need to be considered. Furthermore, because in this context the virtual humans mostly talk and gesticulate, we have a reduction of animation complexity since locomotion and other intricate movements as well as path planning aspects are of minor importance.

However, it is necessary to embed the developed methods into a complete system suitable for dialog applications – not only to simplify the integration of virtual characters into whole 3D applications but also to ease the interaction between real and virtual hu-

Figure 1.3: *Virtual Human Demonstrator ZAMB ("82 Millionen Bundestrainer") at CeBIT 2006, where two human candidates were playing against three virtual characters for winning the position of the coach of the German national football team.*

mans. This implies having building blocks for gestures, speech, and emotions (whilst we especially focus on the dynamic aspects of rendering), as well as adequate layers of abstraction that enable system internal and external use through a unified interface. The proposed concept is coarsely visualized in Figure 1.4. As symbolized by the images in the green-colored layer, the developed algorithms are implemented as scene-graph (SG) nodes, which are made available to external system components via the orange-colored layer, but which also directly can be used for application development.

1.1.2 Simulating Communicative Behavior of Virtual Characters: Open Issues and Challenges

Due to perceptual causality, events that occur within the same perceptual cycle can fuse to a single percept. This fact is not only important for determining the minimum frame rate, but it is even more of importance in combination with human behavior. Thus, human-like communication requires synchronicity and consistency between modalities (e.g. speech with lip-sync and the corresponding gesture or posture) as well as plausibility of appearance and behavior. This requires a lot of functionalities on a lower level of abstraction, which includes the integration of all relevant elements into the visualization and dialog system as transparently as possible.

Interdependence vs. Isolated Applications

Therefore, various demands need to be met, especially in the context of multi-disciplinary collaboration between computer graphics (CG), AI, cognitive psychology, etc. [104]. Likewise, in [319] the authors concluded that "*this is a diverse area of research and this diversity of research is itself a challenge to researchers in this field: each character system has been designed to investigate a particular aspect of non-verbal communication. It is not yet clear if all of this research can be integrated into a single platform [...].*"

Due to the complexity of this topic, first of all this affords manageable, modular system

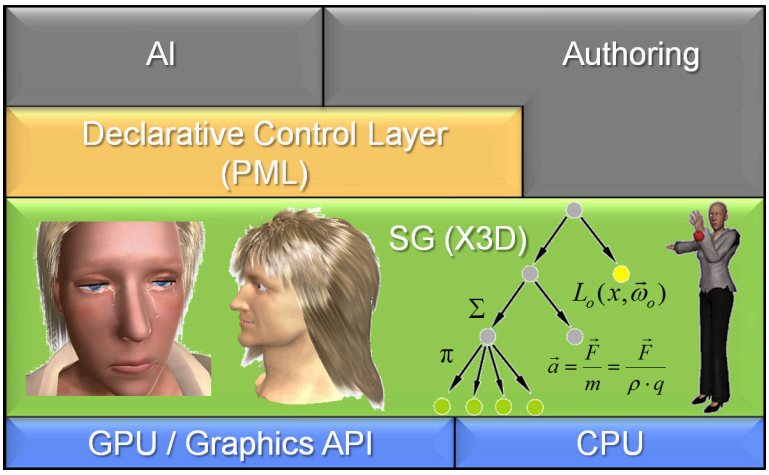

Figure 1.4: *The scene-graph layer (SG, colored in green) provides all necessary low-level functionalities, which can be used directly for application development and authoring or which can be exposed to high-level components (such as AI modules) via the proposed thin, declarative control layer on top of the scene-graph.*

architectures. Currently application development is difficult and inefficient, particularly if a suitable infrastructure concerning tool chain and content creation pipeline, which still requires expensive tools, time, and manpower, is not yet built up. An example here is the games industry, where every company has its own tools and engines. Though the results achieved are often very good, on average the development of a new game title takes more than five years with average development costs of several million euros.

Although the interdependence of different modalities has to be considered, often only standalone systems for specific application types exist. Examples are chat-rooms or a special type of game on the one hand and proprietary, specialized tools as well as isolated applications (e.g. for simulating complex hairstyles) on the other hand. Such components are difficult to integrate, since the used algorithms don't fit together. Generally spoken, it is currently not possible, to integrate all relevant subdomains such that plausible and responsive characters within dynamic environments are available in a manageable manner.

For instance, current commercial systems for interactive real-time avatars like the CharActor system from Charamel GmbH [40] only focus on the character and its animations, including comprehensive animation libraries. Yet, the avatar cannot be embedded into more complex 3D applications but is only available as an overlay. This way the character lacks contextual relevance and the interaction with the overlayed character seems artificial [304]. State-of-the-art animation and game engines in contrast are mostly designed as middleware solution with a C/C++ based API, and because they are targeted at game development, they usually do not provide any form of device abstraction as required for MR/VR applications and to access multimodal input/output interfaces.

Lack of Standards and Accessibility

Furthermore, the used techniques usually are not applicable to other types of applications and there are no readily usable standard components or at least common standards for low- and high-level behavior description. Nevertheless, with recent developments in char-

acter animation and emerging 3D standards like Collada [10] and X3D [336] on the one hand, as well as component-based and service-oriented system architectures and unified interface languages like BML as in the SAIBA framework [318] on the other, this goal now comes into reach. Therefore, designing generic frameworks for interactive agents, including appropriate high-level interfaces and standardizable control languages for specifying communicative behaviors, is another challenge.

Thus, there are ongoing efforts to unify form, interfaces, and generation of multimodal communicative behavior at different levels of abstraction, like planning or realization of a communicative intent [318]. For mediating between these levels of abstraction within the SAIBA framework different interface languages are utilized, like the Behavior Markup Language (BML), which describes verbal and nonverbal behavior on a symbolical level independently from the particular animation and rendering method used. Besides BML several other multimodal markup languages for specifying the behavior of virtual characters like VHML [212] or MURML [193] have been developed (cf. section 2.1.2), either with a human author in mind or for representing expert knowledge that is created at runtime by specialized modules such as a natural language generator [181].

But like BML, they generally only aim at one specific layer of abstraction and are still too far away from fine-grained animation control [123]. Moreover, as already outlined in [104], animation standards such as H-Anim [335] facilitate the modular separation of animation from behavioral controllers and enable the development of the aforementioned higher-level extensions. Yet, it is also remarked that the main problem of H-Anim is the lack of a general behavior and animation control API in the corresponding X3D language binding [336] of H-Anim to ease the development of new real-time behaviors.

However, X3D differs from most other 3D interchange formats in that it also includes behavior and scene logic besides geometry and possibly some meta data. Moreover, although current modeling and visualization systems are often highly specialized and rather sophisticated, they still utilize proprietary formats and methods that are neither in their concepts of operation nor in the supported data formats compatible (e.g., there exist open standards for CAD, though they are rarely fully supported in CAD packages). On the one hand this prevents from a harmonization of 3D assets from different sources and thereby hinders distribution and utilization of 3D contents. On the other hand this can also lead to parallel developments of incompatible and isolated technologies.

Realism vs. Plausibility

Finally, in the area of dialog systems research normally only focuses on face and body animations (especially gestures, mimics, and gaze), following the typical 3-stage model of human information processing (sensory perception, decision/cognition, motor response), whilst ignoring issues concerning rendering and simulation. For example, in the SmartKom architecture [128], the whole presentation component makes at most ten percent of the complete system design, though increasingly powerful computers and graphics cards allow for a more realistic character and scene design.

Typically, in this kind of scenarios rendering and physics lag behind the quality know from computer films and games, since more importance is attached to the dialog itself. In addition, the games and film industry both spend enormous amounts of resources into

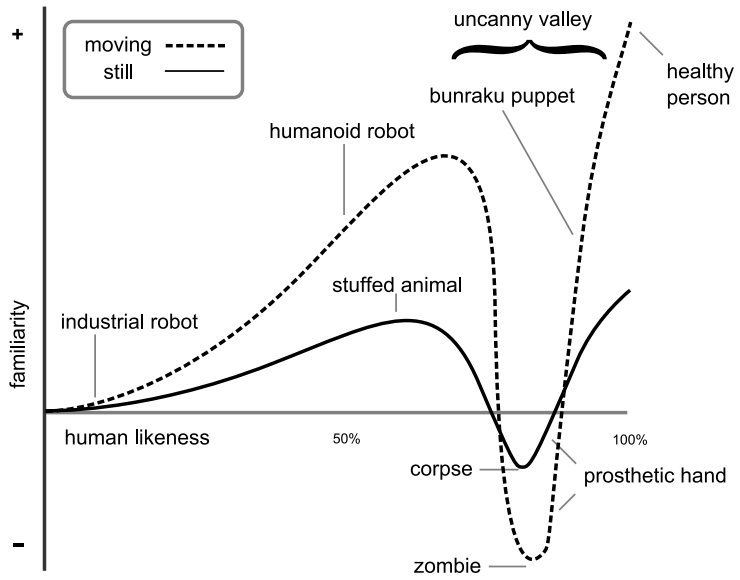

Figure 1.5: *In the so-called uncanny valley the acceptance of anthropomorphic entities is worst.*[3]

content creation, and the latter does not even need to fulfill real-time constraints. Quite the opposite, the algorithms in film production still are fairly slow, thus simulation and rendering of a single frame can take minutes to days.

With visual and behavioral realism, we face the same problems as research on humanoid robots, namely the so-called "uncanny valley", a hypothesis introduced by Masahiro Mori already in 1970 [221]. His hypothesis states that as a robot is made more human-like in its appearance and motion, the emotional response from a human being to the robot will become increasingly positive and empathic, until a point is reached beyond which the response quickly becomes strongly repulsive. Figure 1.5 visualizes this relationship.

Moving robots or virtual characters are even more repulsive than still ones, which explains the frequent use of comic-like and non-human characters in current animation films even today, like in the famous Pixar[4] feature films, where virtual characters are overdrawn in a cartoonish way. In addition, caricaturing an agent can increase communicative effectiveness [32]. But comic-like characters are in general not adequate in the context of Mixed Reality applications, as will be motivated in more detail in section 3.3. However, as the appearance and motion continue to become less distinguishable from a human being's, the emotional response becomes positive once more and approaches human-human empathy levels. This hypothesis holds true for realistic virtual characters, and for convincing results we have to come very close to human-like appearance and behavior [207, 208].

Psycho-physiological Processes

Especially in the context of multimodal dialog systems, in which virtual characters represent the dialog interface and act as personal dialog partners, a reliable and consistent motion and dialog behavior is essential. Here, dialog behavior not only embraces story, dialog management, speech and gestures, but also rendering issues and affective compo-

[3] Compare http://www.androidscience.com/theuncannyvalley/proceedings2005/uncannyvalley.html
[4] http://www.pixar.com/index.html

Figure 1.6: *Combination of affective facial expression (distorted lips and eyebrows) and change in skin coloration for simulating rage (right) compared to neutral expression (left).*

nents. Because human emotions are an important element in a communicative situation, they should also be modeled to achieve plausible virtual characters [104].

The more manlike a virtual character gets, the more people expect emotional behavior. Following Watzlawick [334], nonverbal communication as part of the human behavior always takes place since "one cannot not communicate". In this regard, besides the goal and content levels each communicative act has a relationship aspect. Here, the body language, like gestures and mimics, can reflect emotional behavior and provide information about human feelings. Nonverbal behavior can also support a statement, e.g. with a deictic gesture, or in an extreme case even negate the statement.

Modeling postures and particularly facial expressions of virtual characters both have been subject of extensive research and are getting more and more realistic, for example by using advanced Motion Capturing Systems for creating the morph targets for facial animation (see Figure 2.5 in chapter 2). However, most emotion models that are used for animating mimics, like the so-called Facial Action Coding System (FACS) by Ekman [76], are only suited for classifying expressions resulting from joint and muscle movements, which in computer graphics are represented through mesh deformations.

A more unattended field beyond standard mesh-based animations is the change in color of a face like blushing, which can occur when an emotion is very strong (see Figure 1.6). Thus, also blushing or turning pale, which like gestures and mimics is part of the nonverbal communication, can be important within interpersonal communication. Moreover, except for standard idle behavior, like slight movements as well as blinking or breathing, other unconsciously happening behavior, specifically physiological symptoms like crying or sweating, until recently were mostly left unconsidered.

Although these symptoms of very strong emotions can be a crucial component to simulate realistic affective behavior, for emotionally caused skin color changes and other physiological phenomena there still neither exist approaches similar to FACS nor any other parameterization model. In addition, other dynamic factors need to be incorporated that help to supplement the scene with emotional aspects, e.g. by adding means to simplify camera control, in that subtle effects like weeping are visible, or by integrating methods that allow the character to tear his hair, which likewise needs to be controllable.

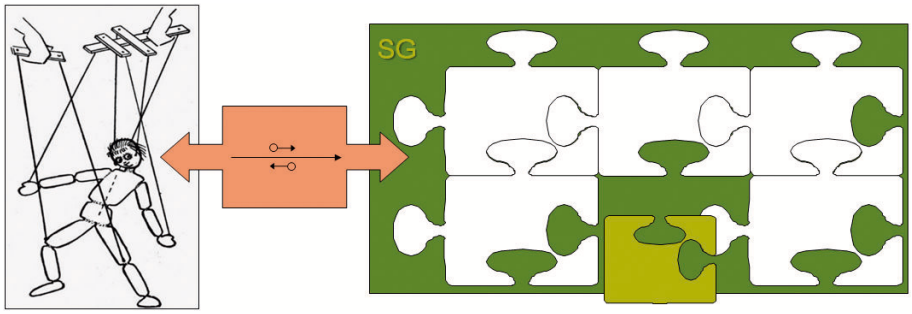

Figure 1.7: *Various building blocks and connector for simplifying character control: high-level control usually is done with AI techniques and symbolized by the puppeteer. Therefore low-level building blocks (visualized as puzzle pieces) are necessary, which all need to fit together and compose the body of the "marionette".*

1.2 Research Focus and Objectives

Important objectives of this work are the provision of suitable techniques and base elements as building blocks for the visualization (Figure 1.7, right) as well as the conceptual design of a model to simplify the integration of virtual characters into interactive 3D applications with dialog-like characteristics. Multimodal dialog systems in general and the previously outlined issues in particular dictate the requirements on the visualization component. A really interactive character requires a high degree of control over its motor control (body, face, voice) and its physiological or externally observable processes, whereas the system must be able to emphasize or modify any of these with suitable camera work, lighting, and other environmental factors.

Though there already exist lots of systems to simulate virtual characters, they are mostly focused on certain subdomains, designed as standalone application using proprietary formats and in-house libraries, or they do not address the demands of interactive and dynamic environments [313]. Especially Mixed and Virtual Reality systems here need to be very variable in that, due to the additional consideration of different input and output devices, application development is rather difficult. In addition, the embodied agent – as a more human-centered and entertaining user interface component – needs to be tightly integrated into larger applications to allow for interactions between users, embodied agents, and the 3D world on the one hand, and on the other hand to avoid missing contextual relevance and that the interactions with the agent seem artificial.

Integration in Standards and Architectures for Dialog Systems

This – and easing scene creation – requires the consideration of integration and standardization aspects. On the lowest level this means the integration of simulation and rendering methods into established visualization techniques like the scene-graph [2]. With the development of modern GPU-based methods (e.g. for simulating hair or tears) there is often a duality of simulation and rendering, which requires an extension of the traditional scene-graph traversal. Further, to minimize the number of concepts one has to think about, the most basic layer that is exposed to the application developer is the scene-graph. Exposing the basic functionality in form of scene-graph nodes results in modular building blocks,

which can also be re-used over different applications and even types of applications.

To ease data exchange and allow other tools or architectures to build upon the presentation component, the integration into existing standards is important. One such example is X3D [336], which is both, a published ISO standard that provides an open architecture to support a variety of domains and, as it is declarative, relatively easy to author. However, X3D neither provides means for advanced rendering effects nor mechanisms for higher level animation and behavior control. Thus, based on the well-defined body structure of X3D's H-Anim component, first of all fundamental elements of dialog systems such as motion blending and speech output are to be integrated. Although these are basic and well-known techniques, most scene-graphs lack this kind of functionality. In addition, to support a more realistic appearance, methods for simulating aspects like hair, skin, and more complex lighting conditions need to be developed and integrated.

Implementing interactions in MR/VR scenarios with the scene as well as the ECA requires supporting special devices and multimodal input and output interfaces. To not reinvent the wheel, the proposed framework and techniques are integrated into the Instant Reality system [135]. Instant Reality is a generic visualization platform that offers a comprehensive set of features including an IO-device abstraction to support VR and AR equally well. The component-based architecture is very flexible and allows the developer to create complex applications with minimal coding, since it utilizes X3D as application description language [22]. The final application can be deployed in various runtime environments like a desktop PC or a multi-screen cluster unit. Integrating the proposed approach into existing standards and frameworks thereby guarantees its availability.

To expose the functionalities of the visualization system to an AI system or user groups like designers, a modular and manageable structure, such as a multi-layered architecture, is needed. Suitable interface and control languages make the visualization layer accessible to those components, which do not act on the polygon level. Instead, fundamental rendering and animation capabilities on the scene-graph level are used as service, since for a dialog it is necessary to control and coordinate the course of animations and events. In this regard, the aforementioned control language is an additional declarative layer on top of the visualization system to make its services available. Figure 1.4 illustrates this relationship. Requirements like flexible animation control, including motion blending and lip-sync, thereby can be directly derived from the needs of higher levels.

As interface and control language, the Player Markup Language (PML), which initially was developed corporately with researchers from the area of AI [183, 204, 181, 156], is utilized and further developed. An important design criterion is the development of a generic PML interface on top of X3D that is also useful for other areas like manual animation scripting, in order to conform to the principles of X3D and system design in general. PML is thereby also a manageable abstraction for application developers and designers and allows to bundle and control different types of actions such as nonverbally expressed emotions and the shader programs that implement effects like blushing.

Considering Dynamic Aspects of Rendering

Since symptoms of strong emotions like blushing or crying can be crucial to simulate realistic affective behavior (Figure 1.8), we will also focus on dynamic skin tone changes

Figure 1.8: *Real-time animation of weeping as another example of affective behavior. Note how the extreme closeup is used to show the characters' emotions in greater detail.*

in order to express strong emotions by taking psycho-physiological processes into account, too. Furthermore, because there are still no approaches or parameterization models similar to FACS [76], a classification model for visualizing emotionally caused skin changes needs to be developed, because such unconsciously happening processes need to be synchronous with voice and motor response and, where necessary, controllable in the same way.

We also look into techniques that can additionally be used to convey certain emotions, which will lead to better immersion as concepts similar to those in the world of film are introduced and made available to interactive (X)3D applications. Thus, for example de Melo and Paiva [56] pointed out, that even the choice of lights, shadows, camera, and lens filters can influence the perception of emotions. The simulation of character-external factors like cameras and realistic lighting conditions is therefore not only important in the area of Mixed Reality, since from a decoding perspective the environment can be used for communication, too. Lighting and camera control can thus help to define, clarify or emphasize the personality, role or interpersonal relations of a character.

Effective camera placement and control in 3D environments is essential to be able to grasp or accomplish a task. However, the virtual camera is still mostly placed using standard 3D navigation methods or rather simple automatic methods. Instead of concentrating on the application, especially unexperienced users are distracted by handling the 3D input device. Moreover, multimodal dialog systems usually require a direct view onto the elements of discourse, since for example the choice of a deictic reference frame depends on a certain viewpoint [45]. Hence, methods to automatically position the camera in interactive 3D applications have come up, which are inspired by cinematography [119]. Therefore, the camera pose is computed based on a description of the intended shot.

Closure

To sum up, this thesis focuses on the definition and development of a generic system for the presentation component of multimodal dialog systems that integrates relevant functionalities and provides them in a manageable fashion, to bridge the gap between symbolic behavior planning and visualization. Therefore, the integration into more complex 3D environments must be possible including the interaction between user, avatar, and scene. In this regard, Instant Reality [135] is a suitable platform that enables a broad range of applications, since it provides a multitude of multimodal input and output interfaces. Availability and efficiency are guaranteed by integrating the proposed techniques into established standards like the scene-graph and X3D [336], which is currently the

only standardized 3D deployment format, and by presenting a standardizable language to specify behaviors, which can be send as a service request message to the player.

Besides this, related subquestions such as camera and animation control are considered, too. However, the focus clearly lies on the graphical representation of virtual characters. In addition, rendering methods are developed, which supplement the application with further dynamic factors. These include models to generate realistic communicative behaviors as well as the whole environment to support the communicative intent and impact. Therefore, unconsciously happening and often emotional processes are also incorporated, which are consistent and synchronous with the character's motor response. Finally, in the case of Mixed Reality applications related real-time rendering methods are needed that are as consistent as possible with the remaining scene.

1.3 Structure of Thesis

This thesis is organized as follows. First of all, related work and theoretical foundations are given in chapters 2 and 3. Then, in chapter 4, we explain the challenges of dynamics related to virtual characters covering play-back and blending of pre-defined animations, online simulation of character motions, and hair simulation. Next, in chapters 5 and 6, we focus on the different aspects of realistic rendering, how standards like X3D must be extended to allow visual effects and the integration of novel visualization algorithms, including skin and emotion rendering, even in Mixed Reality environments. Last but not least, in chapter 7, we describe the aforementioned high-level control language PML, its implementation, and how it interfaces with the scene-graph layer. In chapter 8 we conclude with a summary and possible directions for future work.

The majority of the work described in this thesis has been peer-reviewed. The list of all publications can be found in the last chapter on page 256. These publications tackle various aspects of the aforementioned issues, particularly the six main subquestions resulting from this research, which correspond to the section numbers in the small blue boxes shown in Figure 1.9. The illustration moreover visualizes the coarse mapping from the chapters of this thesis to the corresponding fields of research.

But as can be seen, considering all those disciplines, which generally are all research topics on their own, is a broad field and requires certain containments. However, in dialog systems all these topics are connected with each others, but mostly only dialog and high-level behavior generation, interface languages, and character animation are considered here. Thus, our focus lies on going all the way from high-level models down to rendering issues. In this regard, one main challenge is the connection between low-level graphics on the one hand and high-level behavior control on the other, since there is still a gap between abstract behavior planning and concrete realization [180].

However, to alleviate some of the problems summarized in section 1.1.2 concerning embodied agents in 3D environments and to provide a basis for a sustainable solution, we propose a framework that utilizes the scene-graph concept and builds upon the open ISO standard X3D [336], which is used as the application description language. Thereto, we focus on the different aspects of real-time visualization and animation of realistic vir-

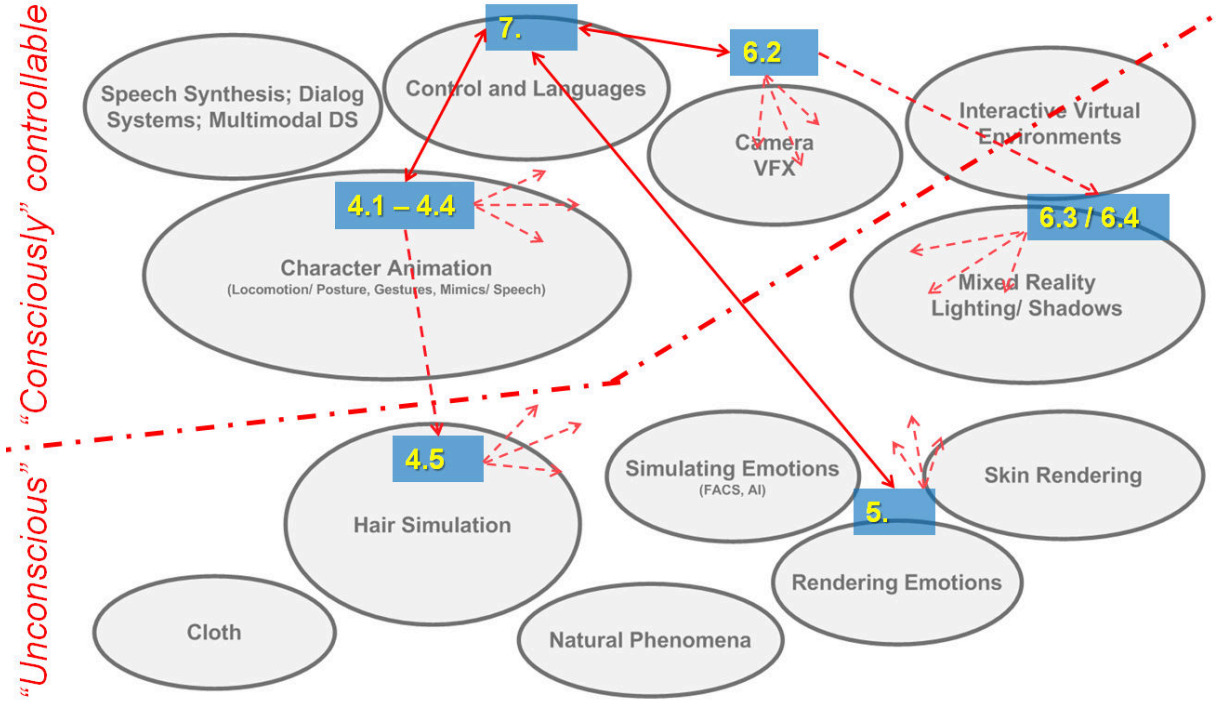

Figure 1.9: *Coverage of fields of research by interconnected components (shown in blue, the numbers refer to the chapters) with respect to related work (shown as bubbles).*

tual characters. One goal is to come up with solutions based on the concepts of current standards and where necessary propose generalized extensions.

Control Layer

In chapter 7 the conceptual framework for controlling and integrating the proposed visualization component is presented. Here it will be explained, how character control is simplified by dividing the various aspects addressed above into different hierarchical layers of complexity (compare page 204). This hierarchy can be roughly categorized into a declarative control layer for behavior description on the one hand (see section 7.2), and an X3D-based execution layer on the other hand. As illustrated in Figure 1.9, we further categorize between consciously controlled actions such as gestures and "unconsciously" happening phenomena [183, 156, 145, 147, 149, 160].

The execution layer is the lowest publicly exposed layer and given in the form of scene-graph nodes. It provides all the building blocks that are necessary to fulfill the requests of the control layer, which is scripted via the interface language PML and responsible for coordinating and synchronizing all behavior. This idea is symbolically visualized in Figure 1.7. In this chapter it is further explained, how the control layer is connected with the execution layer and how it is integrated into X3D and the Instant Reality framework as well as into typical software architectures for dialog systems.

Execution Layer

The other components shown in Figure 1.9 belong to the execution layer and can be controlled directly or indirectly (symbolized by solid or dashed red lines respectively)

via the PML and/or node interface – as aforesaid, the proposed scene-graph nodes are designed such that they are self-contained and beneficial even when used without high-level control. This differentiation mostly coincides – depending on whether "behavior" here refers to either scene behavior in general or character behavior in particular – with the (dashed red) demarcation line between "consciously" and "unconsciously" controllable behavior. Whereas in the majority of cases research deals with the first, we will emphasize on the latter. According to Figure 1.9 we hence contribute to the following research fields:

- Sections 4.1 to 4.4 refer to character dynamics and speech as the apparently most important prerequisites of multimodal dialog systems. Because higher level control requires having the accordant features on the lower levels, we propose a set of self-contained scene-graph nodes for realizing these demands. Algorithmic building blocks and nodes are described that are necessary to fulfill the requests from the control layer to be able to specify and synchronize animations and related events. This includes for instance an audio node for text-to-speech that calculates all relevant input to achieve lip synchronization, as well as dynamic gestures and techniques to generate animations during run-time [183, 156, 145, 147, 149, 160].
- Section 4.5 deals with hair simulation and rendering. A robust and efficient method for real-time hair simulation and rendering is described that is also easily parameterizable. Neighboring hairs are combined into wisps and animated with our proposed simulation system. Simulating hair is important, since despite external forces like wind and gravity the hair has to follow the head movements resulting from dynamically combined or generated body animations. Further, the rendering algorithm adapts recent methods to deliver visual plausibility [158, 154, 156, 145].
- Ungovernable phenomena can either be such adjoint effects that directly follow from the laws of physics and cannot be animated in advance like hair blowing in the wind, or psycho-physiological processes like crying and blushing (see e.g. Figures 1.6 and 1.8), which usually are more or less ignored in current research. Hence, in chapter 5 appropriate techniques for skin and emotion rendering are outlined.

  In this regard we emphasize on simulating emotions like blushing, pallor, and weeping, which occur due to psycho-physiological processes and thus cannot be controlled deliberately. Additionally, we present techniques to simulate sweating and weeping in real-time. Moreover, we propose a parameterizable model to classify and control such manifestations of strong emotions consistently with other behavior. To prove if considering physiological reactions is important for a correct perception of emotions, additionally an experimental study is conducted [155, 187, 148, 162, 338, 159].
- Section 6.2 deals with enhancing camera models and camera control in virtual environments. Besides emotion visualization it is vital to allow scene authors to efficiently use atmospheric effects even in rapidly-designed interactive scenarios. The coordination of graphics and language furthermore implicates several problems for camera control, since e.g. spatial terms can often only be interpreted in terms of a particular deictic reference frame. We thus borrow established techniques from the film area that allow defining objects and object-relations, which the camera uses to automatically calculate its final pose. The proposed camera also includes a model that allows incorporating classical film effects [150, 149, 20, 153, 160].

- Finally, suitable lighting techniques for Mixed Reality scenarios in general are presented in sections 6.3 and 6.4. Especially in the context of MR virtual characters have the potential to represent a natural interface in contrast to standard interaction techniques (where the term MR sometimes is also used to describe conversational situations with real and virtual humans). With respect to dialog systems, it is furthermore discussed how modern rendering techniques can be integrated into X3D, so that the current standard can be utilized for Mixed Reality applications, where it is necessary to integrate virtual 3D objects as seamless as possible into real scenes [151, 89, 88, 262, 160].

1.3.1 Summary

We present a presentation component, or embodied agent realizer respectively, for multimodal dialog systems while focusing onto the graphical representation and especially dynamic phenomena. Our proposed approach offers several benefits, such as sustainability and more efficiency, by means of the integration into well established visualization techniques like the scene-graph, into existing open standards like X3D for more enduring solutions, and into more abstract system architectures as used in the ECA community.

The framework combines important building blocks with a declarative control layer to allow for plausibly reacting virtual characters in interactive environments. This layer serves as an abstraction for behavior description and scripting, whereas the choice of a suitable level of abstraction is essential in that, for one thing, it must be generic enough to be useful for other application areas and, for another, that the interface abstracts away enough from the details to enable a straightforward integration with different systems.

This top-down approach additionally leads to the aforementioned subquestions. Therefore, appropriate building blocks for representing speech and other dynamic factors are presented, which are likewise scalable and multifunctional. Besides the mandatory requirements mentioned first in the bullet-point list above, we will emphasize on unconscious behavior and corresponding physiological and physical processes. An example might be a character that nervously runs his fingers through his hair while slightly weeping. However, research is still only at the beginning. Therefore, further evaluations with expressive avatars in cooperation with psychologists are conceivable as an outlook.

2 Animation and Control Languages

This chapter reviews current research and common techniques in the areas of dynamic high-level character and camera control as well as character animation.

2.1 Interactive Embodied Conversational Agents

As mentioned in the introduction, multimodal dialog systems extend speech dialog systems with additional modalities comparable to human-human interaction, whereas the term modality refers to the five human senses [326]. Similar to face-to-face dialogs, communication is not only based on speech but also on nonverbal communication like facial expressions, gaze, gestures and postures. For the AI components this means a shift from natural language generation to multimodal behavior generation, since the dialog management not only generates voice output but also the corresponding body language. In this regard, an embodied conversational agent (ECA) is a virtual agent that often has an anthropomorphic stature, whose cognitive and expressive capabilities simulate human capabilities, and which is capable of (interactively and usually autonomously) communicating with a user through verbal and nonverbal means [54, 189].

The simulation of virtual characters is a complex topic in research for many years. Such a simulation comprises many research aspects, ranging from digital storytelling, artificial intelligence (AI) to animation and real-time rendering. The ultimate goal is to create virtual human beings, which look, act and react like we do. And even in the field of action driven interfaces virtual humans are gaining more and more importance [144]. Thus, dealing with ECAs requires multi-disciplinary collaboration between different fields of research like AI, linguistics, phonetics, cognitive and social science, psychology, character animation, rendering and simulation and so forth. The main issues from a computer graphics (CG) point of view are character modeling, realistic real-time rendering, dynamic simulations, and natural animations of face and body.

In general, multimodal signals are characterized by their meaning and communicative function on the one hand as well as their visual action (e.g. shown through muscular contraction on a 3D facial model) on the other hand. For example a deictic meaning ("here", "there", ...) maps to a deictic pointing gesture. Moreover, there are lots of interdependencies between all modalities. A verbal utterance for instance requires its temporal alignment with nonverbal gestural behavior etc. Typically, such dialog applications are goal-oriented and communication-driven [104]: on a very high level of abstraction conversational goals and dialog acts are expressed. On their way through the module pipeline, these are then further refined and often enriched with emotions, which for reactive behaviors can bypass behavior planning and directly control presentation (cp. e.g. [181, 180]).

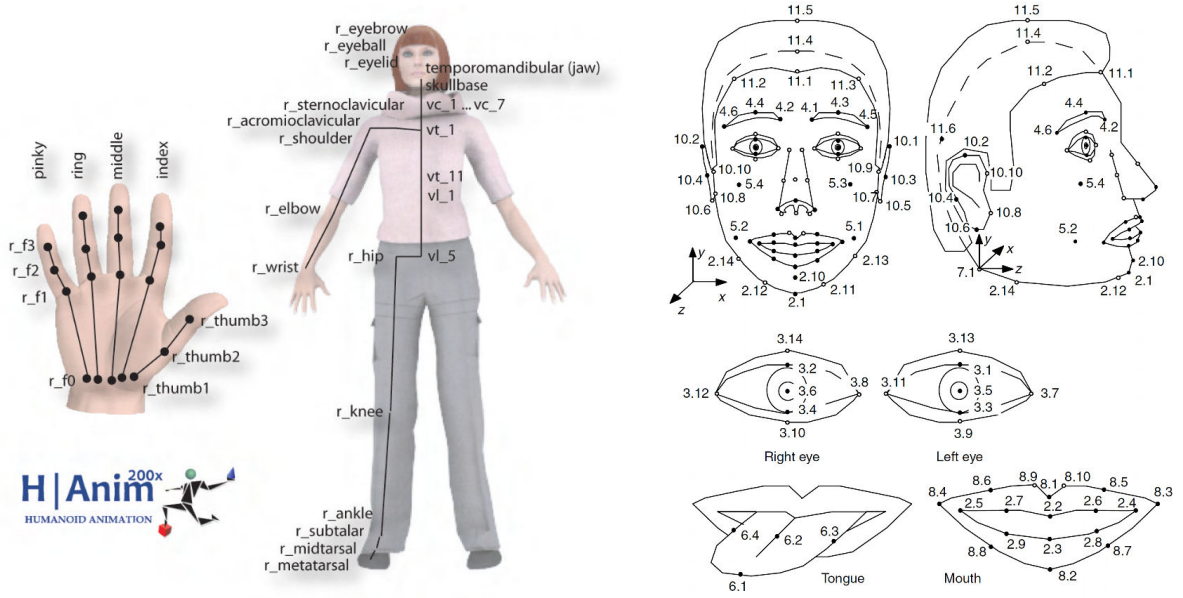

Figure 2.1: *Left: skeleton structure according to H-Anim standard (taken from [73, page 11]). Right: Feature Points in MPEG-4 for modeling facial expressions (cp. [239]).*

Figure 7.7 on page 217 shows an example system.

Usually, each of the loosely coupled modules adds more information until the result is concrete enough for being visualized by a rendering engine. One prominent example is speech: after the TTS (text-to-speech) system has synthesized a sentence, the phoneme durations are known, and a schedule with correct timings, that aligns the spoken sentence with a suitable communicative gesture (including lip-sync), can be build up. Due to the high complexity of every relevant area of research, it is thus important to use standards and to have highly modular system architectures [104, 254]. Furthermore, another issue is the connection between low-level graphics on the one hand and high-level behavior control on the other, since in current virtual agent frameworks there is still a gap between abstract realization planning and concrete presentation [180].

2.1.1 Behavior Control

A character's behavior contains information about the content and expressivity of the communicative act, and it is not only determined by the communicative intention but also by the character's underlying general behavior tendency. Such behavior generally is modeled following top-down approaches like the aforementioned goal-oriented application type. Since nonverbal communication as part of the human behavior always takes place, why "one cannot not communicate" [334], and thereby is an essential aspect of communicative acts, modeling of communicative behavior as such must be handled beforehand on a higher level and is not part of this work, though the visualization component must be able to display this behavior in a flexible way.

Thus, the need for higher level interfaces that allow a more abstract definition and control of object behavior on the one hand, as well as the advancements in real-time virtual char-

acter simulation on the other hand, both call for better mechanisms of animation control and scheduling (beyond the scripts and routes concept of X3D) in real-time frameworks and other middleware solutions that aim at exposing suitable components for an easy and fast application development. Therefore, within the SAIBA framework for interactive agents, three main stages of behavior generation were identified, that are mostly independent of the concrete realization of a character, namely intent planning, behavior planning, and behavior realization [318]. This aims at replacing the previous monolithic or in-house architectures, as for instance used in [144], with a service-oriented software architecture that enables unified and abstract interfaces [123].

Although there already exists a wide variety of commercially available real-time 3D frameworks and game engines, often including powerful level editors, they are neither cheap nor standardized. Besides this, there are many free or open source toolkits and frameworks targeting at advanced character simulation like VHD++ [270, 251], but – if standardized formats are used at all – then mostly only for data exchange but not for defining the run-time behavior. The VHD++ toolkit for example uses the H-Anim standard [335], but only for loading the geometric models. A more comprehensive overview on current animation engines will be given in section 2.2.

Also, work mostly focuses on animation models for dialog systems, such as in [73]. Besides this, application development still affords programming an API in C/C++ or other imperative programming languages, and is therefore not accessible for designers and other non-programmers, who usually design such an application. Moreover, due to our loosely coupled approach, interface languages are better suited than API calls. Thus, in the following we first need to review suitable frameworks and control languages.

By introducing the humanoid animation component (H-Anim) in X3D [335, 336], some of these issues were intended to overcome by specifying the structure and manipulation of articulated and human-like characters. While H-Anim figures are intended to represent human-like characters, they are a general concept that is not limited to human beings. H-Anim figures are articulated 3D representations that depict animated characters. Here, the X3D *HAnimJoint* node is used to describe the articulations of the humanoid figure. The joints are organized into a hierarchy of transformations that describes the parent-child relationship of joints of the skeleton (see Figure 2.1). The skeleton makes up the basic control structures used to animate the character and is also known as *rig*.

However, script nodes and prototyping mechanisms are the only possibilities to achieve and encapsulate some kind of behavior within X3D. This gets even worse for H-Anim figures, which usually are animated with different sets of interpolator nodes storing all key-frames and the corresponding, often dynamically created routes for updating the joint transformations every frame [183]. Figure 2.2 visualizes the X3D standard approach, in which all data is loosely coupled. Although this design works well regarding the reuse of software components, it is confusing, complex scenarios are not manageable, and there is also no information about which interpolators and routes belong to which animation.

X3D *Interpolator* nodes generate a range of linearly interpolated single- or multi-field output values that can be routed to a compatible node field. They differ in the generated value type, for instance an *OrientationInterpolator* generates a series of rotation values that are determined by the current key time and which e.g. can be routed to a *Transform*

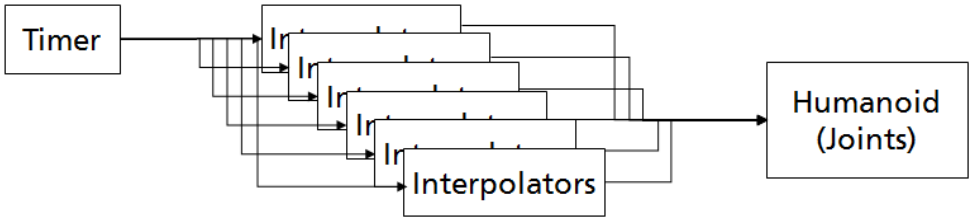

Figure 2.2: *X3D standard approach: HAnimHumanoid and Interpolator nodes with routes.*

node's "rotation" attribute. A similar concept in other APIs is the so-called controller. A controller is responsible for a certain type of animation or behavior, like running or grasping, and can be represented e.g. by a complex script that reacts on sensor input, by the reactions of a physics engine, or simply by a set of key-frames. A toolkit for creating such controllers including physical simulation is presented in [281].

The X3D approach not only leads to lots of data, which is hard to understand and has to be managed, but also requires the application developer to think about well designed prototypes for hiding and encapsulating the data and routing complexity in order to keep the application manageable. The whole concept basically has not changed over the last decade [15, 340], is still labor intense and affords programming skills. Thus, in [327] a method for authoring scenes in the context of networked educational systems, by means of a content production database, a VR modeling language, and so-called VR-Beans for modeling geometry and behavior, was proposed.

The general lack of a unified description for behavior and interactions in current 3D technology with special regards to X3D is also discussed in [50]. Here, the authors propose their "Behavior3D" concept, which amongst others provides (based on ideas taken from SMIL 2.0 [323]) so-called "TimeContainer" nodes that can have a sequential and a parallel realization, in order to overcome certain shortcomings in defining complex animations by supporting better means for synchronization. Likewise, a similar time graph structure for X3D was proposed in [100], but it turned out to be hardly extensible concerning more complex setups, albeit advanced temporal control already is integrated into related standards like MPEG-4 [239, 253].

For developing interactive virtual humans not only the geometric model and some basic ways of animating it have to be taken into account, but also aspects belonging to different levels of abstraction. These range from the shape, as well as key-framed or per joint based kinematics as defined by the H-Anim specification on the one hand, over physical aspects [282] like rag-doll simulation, hair simulation or tissue deformation, up to simple behavior like idle behavior, and finally to an AI driven cognitive layer for handling conversation, personality etc. on the other hand. Because developers have to write many lines of code related to all layers of this hierarchy, which soon gets rather unmanageable for more complex applications, in [132] the authors propose a generic, layered software architecture that allows focusing on the behavioral aspects, whilst providing animation models that also include collision detection and path planning.

In [348] a VRML based system consisting of three layers for animating characters is described. Whereas the lowest layer controls the joints, the middle layer combines a predefined schedule and different joint transformations to skills like "walk" or "open door".

It was shown that a Java3D based implementation not only was faster than a VRML/ Java version but also easier to implement. The highest level was an English-like scripting language for expressing the composition of skills and for hiding the complexity of lower layers. A similar approach is proposed in [130], although in this work the authors use their scripting language (cp. next section) already for composing primitive motions based on operators like 'repeat', 'choice', 'seq' and 'par'.

What is common to all approaches is the fact, that some advanced mechanism for animation control is needed, often with different levels of abstraction for reducing complexity and ensuring portability, in contrast to the simple event triggered mechanisms as provided by X3D. Besides this, if multiple animations shall be combined and displayed, a more advanced mechanism beyond *Script* and *TimeSensor* nodes for scripting and synchronizing all character actions is also needed, what usually is accomplished by means of a scripting language suitable for the corresponding domain, like scripting dance or verbal and non-verbal communication.

2.1.2 Control Languages

As mentioned, realistic behavioral animation of virtual humans is still an open research topic and different kinds of behavioral models depend on the level of autonomy of the character and on whether body and mind are considered independent or not [54]. Behavior definition usually is done with the help of authoring tools or by using scripting and markup languages, which differ mainly in the level of abstraction of the representation they provide, and which often are targeted at multimodal dialog modeling of embodied conversational agents, as in the system proposed by [62].

Here the authors use VHML [212], an XML-based language that was designed to specify the behavior of virtual characters in multimedia applications, and which consists of several sub-languages for describing the character, like GML for its gestures, SML for speech, FAML for facial animation, BAML for body animation, EML for emotions etc. Here, the Emotion Markup Language (see Figure 2.3) was designed to represent the emotional states to be simulated by a user interface or of a human user in a standardized way.[1]

In [54] two XML-based markup languages are proposed, following the idea of a strict separation of mind and body. Whereas DPML represents the discourse plans (the mind's output), a plan enricher then transforms this script into APML (the Affective Presentation Markup Language being the body's input). Likewise MURML [193] allows describing gestures and expressing their relations to speech by defining spatiotemporal constraints and submovements of a gesture stroke. An application example is demonstrated with the anthropomorphic agent Max in [144].

But such languages and behavior models are mostly highly domain specific and incorporate semantic models, like CML [7] that follow an incremental approach and which divides the character into head, upper, middle and lower parts, or the Avatar Markup Language (AML) [196] that consists of AFML and ABML, for e.g. defining style and

[1] EmotionML 1.0 http://www.w3.org/TR/emotionml/
[2] cf. http://www.w3.org/TR/emotionml/#s5.1.3

```
<emotionml xmlns="http://www.w3.org/2009/10/emotionml">
    <emotion>
        <!-- Appraised value of incoming event -->
        <modality mode="senses"/>
        <appraisals set="scherer_appraisals_checks">
            ...
    </emotion>
    <emotion>
        <!-- Current internal state configuration -->
        <modality mode="internal"/>
        <dimensions set="arousal_valence_potency">
            ...
    </emotion>
</emotionml>
```

Figure 2.3: *Shortened example of describing emotion-related behavior with EmotionML.*[2]

path of a walking action. A language that focuses on defining personal avatars in the context of interactive virtual worlds such as Second Life[3], where people often spend a lot of time in configuring their avatar, is ADML (Avatar Definition Markup Language) [237]. It allows defining the avatar identity, concerning communication skills, personality and appearance, independent from graphical formats and virtual worlds to enable the migration between different worlds, but is a bit out of scope here in that this work does not focus on animation and dialog on the one hand, and is targeted at describing avatars modeling real users on the other hand.

The TV program making language (TVML) is described in [120, 140]. TVML is used to create animation scripts with predefined asset like characters etc., to create movements and dialogs, and to control special effects like lighting, camera and music, but it only allows predefined, non-interactive, linear stories and the synchronization of actions is limited. An application where cartoon-like virtual characters are utilized as TV presenters in live television environments is presented in [236], but in this work the focus lies more on speech generation and facial animation.

With SAIBA, another framework for behavior generation is proposed in [189, 318]. As mentioned, in these works a first specification of the control language FML and the communicative behavior markup language (BML)[4] for mediating between a behavior planning and a behavior realization module is introduced. BML describes the physical realization of the intent and defines behavior elements like gestures and facial expressions and also allows specifying constraints for ensuring temporal alignment. By defining an additional dictionary of behavior descriptions, the so-called "Gesticon" [194, 189], the representation language distinguishes between the abstract behavior definition and its concrete realization. Similarly, the Rich Representation Language (RRL, Figure 2.4 shows an example script) is a framework for representing the information that is exchanged at the interfaces between various such modules [247].

[3] http://secondlife.com/?v=1.1

[4] For some more examples see http://wiki.mindmakers.org/projects:BML:main

```
<necaRRL xmlns="http://neca.sysis.at/2002/04/necarrl">
    <participants>
        <person id="tina">
            <appearance character="Tina"/>
            <personality agreeableness="0.8" politeness="impolite"/>
            <domainSpecificAttr role="buyer" x-position="100" y-position="100"/>
            ...
    <temporalOrdering>
        <seq> ... </seq>
    </temporalOrdering>
    <setOfActs>
        <dialogueAct id="v_1">
            <speaker id="tina"/>
            <addressee id="ritchie"/>
            ...
</necaRRL>
```

Figure 2.4: *Shortened example showing a dialog act content specification represented in RRL.*[5]

Because languages like BML usually employ concepts like relative timing (which is resolved with a constraint solver based on sync points [180]) and lexicalized behaviors (where lexicalized means that the behavior can be expressed or categorized by a list of words, like standing, sitting, or lying), recently [123] outlined the need for an additional animation layer, to connect realization planning, where behaviors are processed symbolically via BML, and lower level presentation [180]. BML is also used in other virtual agent frameworks like SmartBody [310] or Greta [225], which both distinguish between realization planning and presentation, though none of them provides a declarative language for animation control [180] – e.g. in Greta, for presentation MPEG-4 scripts are generated.

The aforementioned animation layer is a thin wrapper around the animation engine and situated below higher-level behavior control layers for abstracting away from low-level implementation details whilst giving direct access to the functionality of the animation engine. This functionality is incorporated in the language EMBRScript [123] that describes an animation as a succession of key poses in absolute time. By defining a shader target together with an intensity, hereby also skin tone animation can be achieved. However, for realizing a BML input document, for each specific behavior (e.g. “stroke”) an EMBRScript template needs to be created in advance using a graphical tool [180].

Commercial systems that provide development environments for the integration and implementation of avatar-based human machine interfaces like the CharActor system from Charamel GmbH [40] usually solely focus on the character and its animations. Here, the SDK also includes appropriate voices for text-to-speech and comprehensive animation libraries for typical gestures such as show, welcome or present, which can be controlled via an event-based low-level XML interface. But other scene elements cannot be included. The scripting language used here is supposedly closest in usage to the one proposed in this thesis (see chapter 7.2), albeit there are several differences. On the one hand, as

[5] cf. http://www.ofai.at/research/nlu/NECA/RRL/RRL_docs/RRL_Specification-0.4.pdf

aforementioned, their XML interface is event-based – for example, instead of messages that can propagate events and facts, in their system they rely on various scene triggers like the `<WaitEvent>`, which can wait for a clicked button or the end of an animation.

The language by design thereby does not assume a higher control level. On the other hand, although they do not differentiate explicitly between definitions and actions, there are two types of scripts: one that is mainly responsible for the `<animation_track>` via elements like `<motion>` and `<speak_text>`, and another one that also can have events like `<Show>` as well as character animations, but additionally takes care for defining and loading all scene elements, for setting the background image or for defining active areas. These are partially tasks, which in the here proposed framework are already handled in the X3D-based execution layer to allow a more generic scene setup beyond some predefined assets. Moreover, there is no explicit scheduling element, instead all timing information is given with the help of e.g. `<pause>` tags and `start`/ `end` attributes, whereas all elements can be arranged either sequentially or in `Parallel`.

A similar approach for cut-scene editing in the context of game production is facilitated with *Div2.CSML* (Divinity 2 Cut-Scene Mark-up Language) by Larian Studios[6] for their very recent game "Divinity II: Ego Draconis". Here, CSML is used as scripting language for Larian's dialog designer, a level editor for simplifying content generation workflows that lead up to the production of a cinematic sequence that can be interactive or non-interactive within the concept of a dialog-oriented story script between multiple actors. Thereby, the story writer can define the story structure, write his story and assign a virtual actor to each story- or timeline respectively. Furthermore, a real-time multiple camera system with several lens options is supported, which allow the artists to visualize their ideas and to fine-tune details like animations, audio and lip-synchronization, but certainly can only be used within this special game environment.

Despite this declarative, XML-based approach of behavior control, especially in game and animation engines an imperative paradigm is prevalent. For instance the animation engine PIAVCA [98] provides a Python API for animation scripting. Likewise the 3D modeling package Maya [14] can be scripted procedurally using Python, which recently replaced the former proprietary scripting language MEL Script. Similarly, LindenSL allows scripting Second Life scenarios. Games can be often scripted with the help of Lua[7], a lightweight and embeddable scripting language, which combines a simple procedural syntax with extensible data description constructs. Anyway, the output of high-level dialog engines are descriptive directions rather than algorithmic procedures, which makes imperative languages unsuitable in the context of multimodal dialog systems.

A comparison of common markup languages for scripting and representing virtual characters can be found e.g. in [7], [254], or [237], but in general almost all of them contain semantic knowledge concerning behavior classes, certain body parts etc. and do neither provide any form of control for cameras nor for other scene objects than characters, especially interactive ones such as GUI elements. Although the described languages are conceptually exactly what we need for module communication, they provide the wrong granularity (cf. [123]), leave definition and control of other scene elements such as user

[6] http://www.larian.com/
[7] http://www.lua.org/

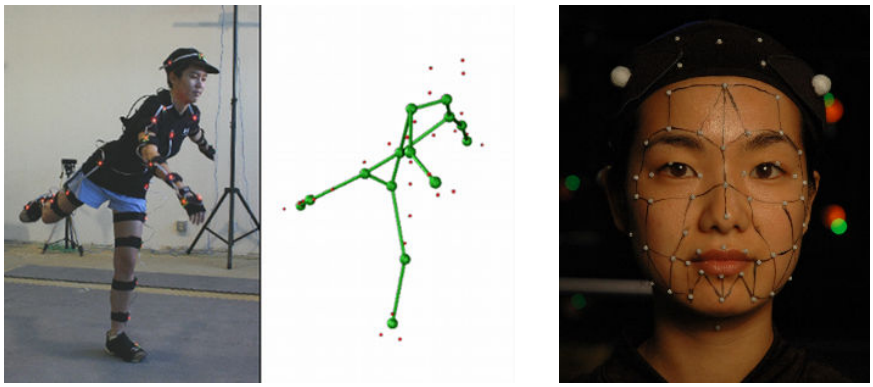

Figure 2.5: *Camera-based Motion Capturing by tracking markers (left) and Facial MoCap.*[8]

interfaces and media aside, cannot deal with behavior resulting from psycho-physiological reactions, and are mostly modeled for a specific domain, wherefore animation engines usually were specially designed or adapted.

2.1.3 Motion Models for Character Animation

To create a new virtual character, first its visual appearance must be modeled. The character geometry can either be created using a 3D modeling software, or alternatively by 3D scanning of real persons. Unfortunately in the latter case there is still a lot of manual work needed to polish the scanned mesh, mainly reducing mesh complexity in order to get it usable for real-time animation and rendering. Albeit there exist model-free approaches for full-body character animation [296], they are usually restricted to facial animation, whereas in the majority of cases body animations follow model-based approaches that expect a predefined model structure.

Thus, the animation of such a polygon mesh is still a problem since the character model first has to be rigged, i.e. a hierarchical set of bones has to be created that is associated with the characters surface representation, namely the skin (what is therefore also called skinning), and which is used for animations only. A good introduction into skin and bones can be found e.g. in [339] or [71]. This modeling process usually is done manually or semi-automatically by highly skilled artists and animators with the help of appropriate modeling packages, and beyond the scope of this thesis. However, di Giacomo et al. [93] discussed techniques to automatically generate a skeleton from a character representation for animating it with bone-based animation techniques.

2.1.3.1 Body and Facial Animation

For character animations such as gestures and postures (i.e. general configurations of the body like sitting or lying), time varying motion data is required, which usually is given as separate channels. Orientation data, which is mostly represented as quaternions or in axis-angle representation, define the angularity of the articulated joints for body

[8] Taken from http://www.moves.com/Psyop face close up.jpg

animations, and position data define their position in 3D space, whereas usually only the humanoid root node (generally the hip) is translated.

While on the graphics layer such transformations are defined as 4×4 homogeneous matrices, where for rigid-body transforms (i.e. angles, distances, and handedness are preserved) rotations make up the upper left 3×3 matrix, in the context of animations rotations are better represented as unit quaternions since they allow for efficient, stable, and constant interpolation of orientations [2, p. 72 ff.]. The relationship between quaternions and the also common axis-angle representation of rotations in $\mathbb{R}^3$ is given as follows, where $\mathbf{q}$ is the quaternion, $\vec{r}$ is the unit rotation axis, and α is the angle of rotation:

$$\mathbf{q} = \left(\vec{r} \sin \frac{\alpha}{2}, \cos \frac{\alpha}{2}\right) \tag{2.1}$$

To animate virtual characters in real-time, mostly the skins and bones approach is used. Here the skin is the geometrical hull (polygon mesh) of the virtual character, which is rendered to the screen and comprises the visual appearance of the character. As mentioned previously, a linked bone system, the rig, is attached to the skin, which is not rendered but used for animating the character. The character's skin can be associated with multiple joints, each with different vertex weights, allowing the vertices to be influenced by all neighboring bones. Each bone thereby influences parts of the skin and a designer only has to animate the bones to create an animation of the whole character [339].

This animation can be done by hand or based on motion capture data. Motion capturing (MoCap) is increasingly used to record natural movements from an actor's performance for the film and games industry (usually with optical methods as shown in Figure 2.5), but it is expensive, requires the use of special suits and markers and is limited to capturing only the skeletal motion of the human body, leaving the dynamics of cloth and hair aside [296]. However, such data delivers the best visual quality though it still needs to be cleaned and optimized by the designer. As humanoid avatars proliferate, being able to reuse MoCap data becomes increasingly important.

But in this regard, retargeting animation data [99] is problematic in many respects. Yet, the preprocessing of MoCap data is critical, since otherwise the acquired data is often unusable. Here, typical problems are filling of gaps, smoothing, value transformations, etc., which are often the most difficult part of the retargeting workflow. Thus, there exist many techniques for preprocessing motion data to make it ready for retargeting. In addition, motion capture data data comes in various types and formats (H-Anim joint nodes for instance are not detailed enough to directly use the raw data), and the amount of generated motion data per joint is enormous. But there is already a whole bunch of software packages for MoCap and data retargeting available, and since this topic is beyond the scope of this research, we won't go into further detail here.

Basically two types of animations can be distinguished, namely data-driven models and procedural methods [98]. Data-driven animation relies on key-frame data like the aforementioned X3D interpolators, where each pose is considered a key-frame. Key-frame animations are defined by a list of samples of key-value pairs, where the key (usually a normalized time fraction $t \in [0, 1]$) is used as index into the input data (usually an

array of 3D positions or orientations). If the current fraction t is between two samples, the output value is interpolated, mostly using linear interpolation for scalars and vectors (with $P = (1-t)P_i + tP_{i+1}$ and $t_i \leq t \leq t_{i+1}$), or spherical linear interpolation (slerp) for quaternions [71, p. 315 ff.] (where $\omega = \arccos(q_i \cdot q_{i+1})$ is the angle between q_i and q_{i+1}):

$$\text{slerp}\,(q_i, q_{i+1}, t) = \frac{\sin\left((1-t)\omega\right) q_i + \sin\left(t\omega\right) q_{i+1}}{\sin\omega} \tag{2.2}$$

A motion sequence can be based on MoCap data or is defined for specific key-frames by a skilled animator using a 3D modeling package like 3ds Max [12] or Maya [14]. Whereas this method allows for realistic and expressive animations, its main drawback is inflexibility. For instance an animation where a character grasps a glass of water on the middle of a table only works for exactly the environment it was made for – if e.g. the glass is moved away, or the table's dimensions are changed, or the character is even placed at the other end of the room, then the character at best will grasp into the void. Some more advanced techniques to alleviate this problem are described in the following section.

Procedural animation methods like inverse kinematics (IK) [311] tackle this problem by algorithmically generating animations on the fly. The basic idea is to determine a joint configuration of a given kinematic chain (e.g. an arm) such that an end effector (e.g. the hand) achieves the desired pose, by simultaneously minimizing for biomechanical constraints of the limbs to obtain more realistic poses. Because in the general case no analytic solutions exist, usually numerical methods (or a hybrid approach also utilizing analytic methods) are used, where ambiguities are resolved via energy minimization.

One famous algorithm that solves the IK problem through optimization by minimizing the distance between the target as well as the end effector and joints is CCD (Cyclic Coordinate Descent) [35], since it handles any number of joints and is rather efficient to process. By looping through all joints of the kinematic bone chain from its end effector to the root, each joint is optimized such that the end effector is moved as close to its target position and orientation as possible, whilst conserving the initial distances between all joints and angle limitations. This loop is repeated until a solution is found through convergence or a given iteration limit is reached.

Other issues of IK are the high number of degrees of freedom (DOF) regarding the respective configuration space and the inability to produce natural motions. With "action capture", in [143, 5] the problem of goal-directed motions is tackled at a higher level, in that a certain animation given by conventional MoCap is always considered as a kind of skill (based on imitation learning), including the interaction with scene objects, like grasping a ball or hitting a switch of a device, within a VR setting.

In addition, the physically-based ragdoll simulations known from games are mainly suited for motions like suddenly falling down (especially when dying) that are not consciously controlled. But often these motions appear rather unnatural since only the skeleton is simulated without considering the nervous system, muscle stiffness and so on. There exist also approaches that aim at combining ragdoll simulations with standard animation techniques to achieve more plausible results [282].

Facial animation usually is done with Morph Targets or Blend Shapes respectively [4, 71], which are complete facial expressions that can be blended into each other by setting their blend weights over time. The morph targets, including a neutral expression, can be considered the target states of a modeled face, e.g. a smiling one, one with closed eyes for blinking, one saying 'a', and so on. Each of these n states is taken as a base vector of an n dimensional space spanning all possible combinations of point sets. In order to get valid linear combinations, the coefficients (weights) of these data points (i.e. sets of expressions) must sum up to 1, which is thereby in fact a convex sum.

In contrast to this, in the MPEG-4 standard facial expressions are modeled by modifying certain Feature Points (FP) as shown in Figure 2.1 [239, 253], which basically corresponds to the Facial Action Coding System discussed in section 2.1.3.3. Here, animations are achieved by using the very low-level Facial Animation Parameters (FAP) that define, which FPs, including their assigned vertices, are moved in which direction. MPEG-4 also defines high-level parameters like visemes and expressions. A viseme denotes a specific mouth shape that can be used to describe a particular sound [11]. It is the visual equivalent of a phoneme in spoken language to allow for lip-sync, whereas 14 discriminable visemes are defined in this standard (e.g. a special phoneme-viseme mapping for German is presented in [11]), as well as the six emotional expressions visualized in Figure 2.6.

A prominent and early researcher in the area of character animation is N. Badler [16], who also was one of the first to represent facial animations with the help of controlled muscle movements [248] based on the Facial Action Coding System (FACS) described in section 2.1.3.3. Since then Thalmann et al. dealt quite comprehensively with virtual humans in general, starting with subdomains such as offline hair simulation [200, 53], wrinkles and skin [353], over idle behavior generation [75] up to crowd simulation [309]. Besides this, real-time character animation gets increasingly important for game development [71].

2.1.3.2 Motion Planning and Synthesis

The aforementioned term "configuration space" (C-space or $\mathcal{C}$) stems from robotics and motion planning [42, page 39 ff.], where a configuration describes a certain pose of an articulated system (e.g. a robot), and the configuration space is the set of all configurations, while the motion itself is considered a path resp. continuous curve c in the free C-space $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \cup_i \mathcal{C}_{obstacle_i}$, whereas the number of DOFs denotes the dimension of $\mathcal{C}$. Generally, motion planning is distinguished from path planning in that the latter is parameterized by time t, i.e. in this case $c(t)$ is a trajectory.

Hence, in [261] motion behavior is divided up into three hierarchical levels – action selection through higher level goals, steering, and locomotion. This work focuses on the second level, path determination for autonomous agents (or non-player characters/ NPCs as they are called in games), by describing a set of steering behaviors such as "seek", "path following" and "obstacle avoidance", leaving animation aside. In contrast to motion planning that only refers to high-level goals, motion synthesis denotes the generation of the low-level details of a movement [244].

To achieve flexible high-level character control with natural movements, in [302] therefore a multi-level (yet offline) approach was presented. First, at the highest level, path

planning takes place, e.g. by simply using A^* or, as proposed in [302], by utilizing probabilistic roadmaps (PRM), which randomly sample $\mathcal{C}$ for finding configurations in $\mathcal{C}_{\text{free}}$ that make up the roadmap (cf. [42, p. 202 ff.]). At the second level, the resulting path is approximated by a composition of motion clips obtained from searching the motion graph (described below), which requires a preprocessing step that annotates all clips for finding possible transitions to build the graph. The third and lowest level deals with adjusting the motions, e.g. by blending motions together, to follow the path at the best.

In [33], the authors try to simplify the process of modeling skeleton based humanoid animation for creating sign language animations by proposing a visual modeling tool, where complex animations are created from sequences of simple animations by building a linear transition between them. This leads to acceptable results, if the motions are not too different and no penetrations occur. To be able to blend between more distinct motions, blending in games usually is achieved via Blend Trees (compare e.g. [72]), which are setup manually and define possible transitions between different motions.

Basically following the same idea, research lately has focused on motion generation based on data-driven methods to produce human motion from collections of example motions. This strategy is called synthesis-by-example [244], though a lot of example motions are required here. One prominent approach are motion graphs [192, 201]. They can be seen as generalization of blend trees and are directed graphs where all nodes correspond to motion fragments taken from motion capture data, in order to obtain realistic human motion including the subtle details of human movement, that are not present in procedurally generated motions via e.g. inverse kinematics [98].

Motion synthesis is done via graph walks that require a certain connectivity, which is not given per se, because usually no two motion fragments are sufficiently similar [192]. Therefore transition motions that seamlessly connect two motion fragments have to be created and a set of candidate transition points have to be detected. Figure 4.8 (page 108, left) exemplarily shows a very simple motion transition graph, where the nodes represent the motion clips and the edges correspond to possible transitions. Additionally an animation controller is needed that assembles motion fragments based on the current state and input. Although these methods lead to convincing results, even for on-line motion generation in game scenarios, they still require preprocessing, manual work, are computational expensive and high memory consumption is still an issue [217]. Thus, most approaches are still targeting at off-line motion generation such as the one presented in [268], where a modified A^* is used for searching the motion-clip graph.

Complementary to the motion graph approach are parameterizable motions [265, 290], where the focus lies on generating parameterizations of example motions such as walking, jogging, and running [242, 241], whereas a motion is represented by its parameter vector p, with e.g. $p = (velocity, style, path)^T$. In Figure 4.8 (right) an example is shown. Again, the goal here is to synthesize new motions based on motion captured animation clips resp. example motions. But these models still also have to deal with the same problems as the previously mentioned motion graphs.

In general, parameterizable motions tackle the problem of design freedom, since customizing given animations to other characters, motion paths, styles, or environmental parameters per se is hardly possible. In this context, van Welbergen et al. recently surveyed

real-time animation techniques in terms of their motion naturalness and the amount of control that can be exerted over this motion [317], though this report focused on animation systems in general without considering high-level control.

The idea is to be able to change motion parameters like velocity, emotions, or path, which in turn changes the properties of a set of motion captured animations while retaining impression of realism. By building up a meta model of motion, certain properties thereby are parameterizable [265, 290, 241]. Main issues e.g. in the work of Park et al. [242, 241] are dynamic time warping (DTW) for temporally aligning the example motions, weight calculation, and posture blending. Following [290] the weights of the motion cycles are obtained using a cardinal basis function for each of them, consisting of the sums of a linear and a radial part over all parameters. This in turn determines, in how far a motion contributes to inter- and extrapolating the new motion. In this regard it is important to keep track of key poses like "foot contacts the floor" to avoid artifacts such as foot sliding and penetration when calculating the transition motions.

2.1.3.3 Facial Expressions and Emotion Theories

People not only are influenced in their thinking and actions by emotions but also are attuned to recognizing human emotion [273, 314, 113]. Thus, virtual characters that display emotions are critical for believability, because emotions create the difference between robotic and lifelike behavior. Emotions are a physical and psychical reaction to external and internal factors and usually cannot be controlled consciously. Emotions are reflected in the facial expressions, gestures, voice, diction, and a person's behavior in general, but the intensity in which emotions are visible differs between persons. There exist various emotional models. But research is still divided about which emotions are considered to be basic emotions or even about the question how exactly emotions can be defined.

In their well-known anatomically oriented Facial Action Coding System (FACS) Ekman and Friesen [77] distinguish the following six emotional expression groups in conjunction with their corresponding geometric "deformations": surprise, anger, sadness, disgust, pleasure, and fear (see Figure 2.6). In this model emotions are discretely represented by universally recognized basic emotion prototypes. Complex emotions can be composed by combining several basic expressions appropriately, though according to [272], it is not possible that a human being can have two basic emotions at the same time.

In [77] the authors further describe a set of minimal Action Units (AU) that correspond to all visually discriminable basic movements, which map to individual muscles, multiple muscles, or even joint motions. To represent a certain facial expression, generally several AUs (e.g. "nose wrinkler" or "upper lid raiser") are activated simultaneously. Temporal information is not yet provided in this model. Hence, other specific values such as intensity and time have been integrated, too [78]. A quite elaborate discussion on emotions in the context of simulating the expressions of virtual characters is given in [319]. However, here the focus lies on discussing the linguistic, socio-scientific, and psychological grounds, whereas rendering and implementation issues in general are only lightly touched upon while psycho-physiological reactions like blushing or crying are not mentioned at all.

[9] Taken from http://www.fhv.at/edu/diplom/im/showroom/emotion/

Figure 2.6: *Stylized facial expressions of six basic emotions following Ekman. From left to right: surprise, anger, sadness, disgust, pleasure, fear.*[9]

The FACS model is used for creating a virtual character's muscular expressions or morph targets respectively, but disregards changes in complexion, though it was recently shown [66], that reddening cheeks can soften others' judgments of bad behavior and also help to strengthen social bonds. A system for 3D chat agents that also includes facial color changes for providing emotional content was presented in [38]. Color changes were considered as an additional animation channel like facial expression and eye movement, but the authors neither aimed at realistic rendering nor did they propose a model for mapping color changes to specific emotions.

The psychologist Robert Plutchik [249] developed a psycho-evolutionary theory showing eight primary human emotions and extended Ekman's model by also adding two other emotions: acceptance and anticipation. Therefore, Plutchik proposed a three-dimensional model, which describes the relations among emotions. A vertical dimension represents intensity, and both horizontal axes denote affect arousal (active vs. passive) and affect valence (positive vs. negative). A circle in the middle represents degrees of similarity among the emotions, e.g. grief (high intensity), sadness (medium intensity), and down (low intensity). This relationship is visualized in Figure 5.9 (left) on page 147. Whereas Ekman's discrete model is frequently used for displaying emotions, dimensional emotion representations are often used for emotion recognition.

In their OCC theory, the psychologists Ortony, Clore and Collins [233] mention three reasons for emotions: situations that trigger the emotions, persons who feel these emotions, and the appraisal of the situation by this person. The OCC model thus belongs to the class of so-called appraisal theories of emotions, which here can only be caused cognitively through a subjective and continuous evaluation of the environment. It is a structured explanatory model for all emotions, where emotions are defined as valenced reactions on activities, events and objects. Thereby three main groups of emotions are differentiated: those triggered by an event affecting a person's goal, an event affecting a principle or a standard, or by concrete or abstract objects. Because emotions including different intensities can be classified based on rules and decision trees, the OCC model is also commonly used in AI for triggering emotions [21].

2.2 Overview of Virtual Character Systems

In [141], it is shown with the example of training clinical examination interview skills, that a virtual human experience can be as effective as a real human experience in interpersonal

skills education. Furthermore, the virtual humans should be embedded into the application to provide contextual relevance and to avoid that the interaction with the character seems artificial [304]. Often tools for creating digital stories are usually quite complex and not intuitionally to use, besides the fact that a connection to graphical components is only rudimental. To alleviate this problem, in [68] a multi-tier approach for augmenting MR environments with conversational, pedagogical agents was proposed.

But as mentioned, the main focus of current multimodal dialog systems (like the already discussed virtual agent frameworks SAIBA [189, 318], SmartBody [310], and Greta [225], as well as e.g. RealActor [37], the EMBR architecture [123, 180], or the ECA Max [144]) concerning output modalities lies on gestures, mimics, and speech, yet rendering and psycho-physiological processes are widely neglected. An overview on current character engines (from a bottom-up point of view), including a comparison concerning skeleton setups, animation generators (e.g. data-driven vs. procedural), and control, was recently given in [98]. Based on the results of Thalmann et al. and for integrating further research the vhdPLUS development framework was developed, which is based on the VHD++ framework [251]. It is a service-based middleware solution and thereby extensible on the code level, but many features like cloth and hair simulation are not public and moreover the toolkit is not suited for non-graphics people.

Likewise Egges' research [73] is based upon the VHD++ framework. Here, the focus lies on the development of an animation model for interactive dialog applications, paying particular consideration to mimics, gestures, and idle motions. The model combines several approaches, like motion captured animations with procedurally generated ones, in a consistent and natural manner, while also considering emotional states. For one thing, this is achieved with the help of a principal component analysis (PCA) that is applied to an exponential map representation of rotations as proposed in [3] for real-time animation manipulation, and for another, by designing a framework for blending various concurrently running face and body animations. This engine and the underlying motion-synthesis-from-analysis technique are also described in [210].

A similar thesis was presented by Tümmler [313]. He stated, that in the area of virtual characters a lot of good but isolated applications exist, which normally can be hardly combined to a total solution and in practice often have to be re-implemented. Though here the main focus lies on avatar modeling and real-time animation of avatars on the one hand and their integration into a rendering system for CAVE environments on the other hand. In addition, the thesis deals with preparing the toolchain for content creation in greater detail, but despite the primary statement concerning isolated applications only proprietary file formats like FBX [13] are utilized. The same goes for the rendering and animation component, for which (based on open-source systems like OpenSceneGraph [231] for rendering and Cal3D for animation [34]) an own runtime system was developed, whereas scripting was realized with the help of a Lua interface.

Rather similar to dialog-like systems, at least on the rendering and animation side, are virtual community environments on the web. Figure 2.7 shows screenshots of three contemporary 3D online worlds (from left to right: Second Life, vSide[10], and Vivaty). This kind of online communities aim at "socializing" but usually serve commercial purposes

[10] https://www.vside.com/app/start

Figure 2.7: *Screenshots showing three commercial virtual online communities with 3D avatars (cp. text below for more information). From left to right: Second Life, vSide, Vivaty.*

and mostly allow for including additional user generated content. As can be seen in the images, vSide (in the middle), whose target audience are teenagers, avoids the uncanny valley effect by utilizing non-photorealistic, comic-style rendering.

Vivaty, which shut down in April 2010,[11] but then was bought by Microsoft (probably due to the upcoming 3D Internet hype caused by the introduction of WebGL [175] in late 2009), was quite interesting, because it is based on the X3D and H-Anim standard [336, 335]. Conceptually, these online worlds are often comparable to a so-called MMORPG (massively multiplayer online role-playing game), like "World of Warcraft" by Blizzard Entertainment. But in contrast to dialog systems, where generating multimodal dialog behavior is prevailing, those systems have more in common with distributed virtual environments (DVE, an example is e.g. Avocado [312]), and the avatars, which are either dumb bots or controlled by the user, only provide a basic set of behaviors.

2.2.1 Modeling and Animation Tools

There exist a variety of commercial products for character and animation creation. One tool, which seems fairly straightforward to use, is Curious Lab's Poser.[12] The software offers characters ranging from comic-style to realistic including appropriate motion libraries. Furthermore commercial companies offer additional character models at low price. Slight adaptations of the characters can be easily carried out in Poser. Most characters offer a wide range of morph targets (for visemes and emotions) which can be simply applied to form more complex animations. Even animating a character by hand is straightforward.

Unfortunately its export options are very limited and erroneous. For example to use the created animations and characters in 3ds Max the complete model must be adapted and revised. The H-Anim output only delivers full-body morph meshes based on X3D *CoordinateInterpolator* nodes, which is not suitable for real-time interactive scenarios. But Poser features a Python-based API, which allows accessing nearly all internal data structures – unfortunately except for the correct vertex weights as needed to export the mesh and animation data into H-Anim format.

Other 3D modeling tools, such as Maya [14], 3ds Max [12], Cinema4D [216], and Blender, are not specialized on characters but allow creating all kinds of geometry and dynamics.

[11] http://www.gamasutra.com/view/news/27916/Vivaty_Shutting_Down_3D_Virtual_Spaces.php
[12] http://poser.smithmicro.com/poser.html

These are either directly rendered in an offline process using those tools (like for film production), or – if possible, as in the case of meshes and key-frame animations – they can be exported into formats suitable for interactive real-time applications. One such format is COLLADA, an XML-based format that aims at establishing industry-standard 3D interchange to overcome proprietary ad-hoc modules for importing and exporting 3D content with the help of a unified Collada-based pipeline [10].

Similarly, the X3D specification [336] defines several profiles (sets of components) for various levels of capability such as X3D Interchange, X3D Interactive, and X3D Immersive, yet in contrast to Collada, X3D includes a full runtime, event and behavior model. X3D is therefore much more than a simple exchange format, though Collada additionally provides support for grouping animation clips as well as for contemporary rendering features such as effects/ FX, e.g. by introducing a `<pass>` element that can refer to various render targets. Another asset exchange format is Autodesk's proprietary FBX file format [13], which is widely supported in DCC tools like 3ds Max, though a big problem is the fact that there is no open specification available and the format itself is being constantly modified.

For comparison and evaluation, in the following some presently common animation engines are discussed. EmotionFX[13] is a commercial real-time character animation system that can be used for a 3D engine, game etc. The C++-based SDK is designed to take advantage of hardware that allows parallel processing, and the corresponding artist tools include full body skeletal and facial animation support, motion mixing, lip-sync, real-time motion retargeting, inverse kinematics (especially to plant feet on uneven terrain and to prevent foot sliding), ragdoll support, a so-called jiggle controller that allows simple physics to be applied on bones, as well as exporters for 3ds Max and Maya.

Granny3D [103] is a toolkit to support the creation of interactive 3D applications. The toolkit is specialized on dynamic animation of virtual characters and also provides a blend graph [72] editor that allows an artist to create blend operations graphically. Cal3D [34] is a free graphics-API-independent, skeletal-based character animation library, and supports combining joint-based animations through a C++ mixer interface. IKAN, which is available from SourceForge, is a C++ inverse kinematics library that can be integrated into other software systems. It provides a set of inverse kinematics algorithms suitable for an anthropomorphic arm or leg, utilizing a combination of analytic and numerical methods to solve for position, orientation, and aiming constraints as described in [311].

Endorphin (developed by the company NaturalMotion[14]) is an authoring and dynamic motion synthesis system utilizing AI techniques and biomechanics to simulate human behavior. The characters are not directly animated by the user, instead he defines the code of behaviors, though simulation parameters can be influenced. The corresponding animation engine Euphoria is a toolkit used to simulate a character at runtime and is available for PC and most typical gaming platforms.

A software system aiming at embedding virtual humans into real-time simulations (especially military scenarios) is DI-Guy[15], which comes along with a rich library of characters

[13] http://www.mysticgd.com/site2007/
[14] http://www.naturalmotion.com/products.htm
[15] http://www.diguy.com/diguy/

and animals as well as corresponding motions. Havok[16] provides SDKs mainly related to the physics of a game such as falling boxes, but also supports other features like motion blending, inverse kinematics, or authoring of character behavior. A comparison of current animation toolkits can also be found in [313, page 45]. But in short, all of them use proprietary formats, define their own content pipelines etc.

2.3 Camera Control

Generating multimodal output such as natural language and graphics involves the coordination of all relevant modalities. Typically such systems are developed in the domain of education and training and need to address the problem of coordinating the choice of vantage point from which to display the objects that are described or referred to linguistically [45]. For example, a direct linguistic reference to an object, e.g. a button in the GUI "over there", whose functionality is going to be explained, usually requires that this object is visible or at least not fully occluded in the shot. Thus, multimodal dialog systems usually rely on the use of default viewpoints from which unoccluded views of the elements of discourse can be achieved [44].

Beyond such simple settings, the coordination of graphics and language poses a number of problems for camera control, because the semantics of spatial terms can only be interpreted by reference to an appropriate perspective [45]. E.g. descriptions involving spatial prepositions assume a particular viewpoint, on which the choice of a deictic reference frame depends. Moreover, camera control and lighting allow defining or emphasizing a character's personality, role or interpersonal relations. But with a few exceptions up to now camera control has received little attention in computer graphics. Since capturing and communicating information by deciding where to position and how to move a camera in relation to the elements of the scene is a major concern of cinematography, Christie and Olivier [44] presented an overview of automatic camera control while also analyzing main difficulties and challenges. Yet lenses, filters, and other visual effects, which are important for expressing moods or directing attention are left aside, though with recent GPUs they can be implemented rather efficiently in hardware.

2.3.1 Standard 3D Navigation Types

Typically, in 3D applications a few suitably chosen viewpoints and some camera animations are defined by the application developer, which usually is achieved by navigating through the scene and specifying camera positions and viewing directions such that important regions of interest are shown. Navigation then means, to change the camera pose interactively according to the given interaction type (e.g. pan, zoom and trackball navigation for examining an object, or a fly- and walk-through navigation). Thus, low-level graphics APIs like OpenGL only provide means for calculating a simple "lookAt" transformation and in [286] it is additionally explained how to compute the field of view for framing the full object of interest. This is almost the same for higher level middleware

[16] http://www.havok.com/

APIs like e.g. OpenSG [232], which provides special "Navigator" classes for the previously mentioned basic navigation types.

The ISO standard X3D [336], which allows defining scene description and runtime behavior of a 3D scene-graph on a descriptive level by editing XML instead of C/C++, additionally specifies a special "lookAt" mode (that can be set via the 'type' field of the *NavigationInfo* node). It is used to explore a scene by navigating to a particular object by selecting it: In this case the viewpoint moves to some convenient viewing distance from the center of the selected object, with the orientation set to aim the view at the approximate center. The InstantReality framework [135] that builds on top of X3D additionally provides a "showAll" mode, which is especially useful for a first orientation or when the user got lost. In [46] the Alice system is presented, which allows describing the time-based and interactive behavior of 3D objects more declaratively. Rather than operating in XYZ-coordinates, here all commands, including camera placement, operate from an object's local coordinate system to ease usage.

Primarily the most crucial aspect concerning Virtual Reality (VR) is *presence*, for which highly immersive environments and special VR-devices are needed (like space mice, finger tracking or cyber gloves – the latter also demands knowledge of special metaphors from the user), which are still uncommon, usually only available in one size, and in addition quite expensive. Navigating through a 3D view of a building or landscape means controlling the virtual camera's position and orientation. But often people still have problems to keep control while navigating with common VR interaction devices, and thus, they usually have difficulties to fulfill typical reviewing tasks, like moving the camera around a 3D object while keeping it in focus, or moving it along a building's facade.

With e.g. a space mouse a user has all degrees of freedom he might need to take a look around. But controlling the camera is still an issue and one has to be experienced in using this device. So, instead of simply navigating and concentrating on the review process, users are distracted by handling the interaction device [267]. Another concept are so-called mouse gestures that are used in web browsers or in multi-touch environments, where the user can interact e.g. by moving one or two fingers on the table-top [176].

In [153] novel techniques of controlling 3D content using multi-touch interaction principles for navigation and virtual camera control in the context of X3D are introduced. The virtual camera can be controlled by simply positioning the fingers onto the multi-touch table's surface by utilizing a specifically for multi-touch interaction designed X3D pointing sensor [152]. One first finger touching a map or blueprint controls the position of the camera. Using then a second finger defines the direction to look at. Mapping camera parameters onto the user's fingers supports him performing navigation tasks, and is, unlike using artificial interaction devices or even the mouse, a very intuitive and direct means for navigating in a virtual environment.

A similar approach for navigating in 3D, but based on certain gestures, is proposed in [110]. However, the presented gesture sets are limited, and an unmanageable amount of gestures, which might confuse the user who must remember them should be avoided. Furthermore, it also depends on the application, which type of camera shall be used: for instance in Augmented Reality applications the virtual camera directly follows the real camera pose as determined by the tracking system.

2.3.2 Cinematographic Principles

Cinematography is the art of motion picture making, communicates the emotional state of a character, and includes aspects such as camera filters, lenses, framing, camera movement etc. A movie is made up of sequences of scenes that consist of several short shots (i.e. a continuous view that is filmed with one camera without interruption) [182]. Here, it is important to maintain spatial and temporal continuity. Over the last century, filmmakers have developed a set of guidelines that allow involving the viewer in a story, and which nowadays are taken for granted [119].

One such rule is the so-called "line of action" or "line of interest" (see Figure 2.8), which is either formed along the direction a single actor is facing or moving, or by the line connecting two interacting actors [119]. The camera placement is specified relative to that imaginary line, which must not be crossed with any camera once it has been established in order to avoid confusion for the viewer. A common placement is for instance the so-called apex view (the camera in the middle shown exaggeratedly near in Figure 2.8, right), which frames two actors such that each of them is centered on one side of the screen [43, 121], or an over-the-shoulder shot that already requires semantic knowledge.

Cinematic terminology is derived from character oriented shot compositions, such as over-the-shoulder shots or close shots. These terms require a semantic rather than just geometric representation of objects. Furthermore, translating cinematographic notions into some sort of controller nodes is problematic, because even the seemingly simple notion of a shot encompasses many possible solutions [45]. However, high-level tools based on cinematic constructs can represent an advance over the existing keyframe-based methods.

Another important aspect concerns framing the scene, whereas generally the size of an object, mostly a human actor, is proportional to its present importance. Thus, some standard shot sizes like "full close-up", which shows the head and throat, or "full shot", which shows the full body by covering the whole screen vertically, have been established [43, 182]. Furthermore, there are also guidelines concerning the location of an object in the frame, which can be described by the so-called "rule of thirds" (see Figure 2.9 for two examples). Changing the sizes of an object on the screen usually implies changing the camera distance. Therefore, occlusions can occur, which can be alleviated by modifying the camera's field of view instead. The target locations can be further modified by changing the camera angle relative to the line of action and floor plane [119]. In this context, the problem of editing is distinguished from viewpoint planning [45].

Medium and close shots are classically used in dialogs in order to show the expressive parts of the body (gestures and mimics), whereas medium close-ups display the characters face, allowing to following the speech and to capture emotions. As can be seen in Figure 6.2 (v) on page 60 even subtle effects like tears can be framed. Extreme close-ups can be used to emphasize the characters attitude or reaction, whilst focusing on the face and eyes. So for example in a dialog, the height of the eyes is utilized to establish relationships between characters: eyes at the same level deliver a sense of balance between them and support the narrative in that way and vice versa. Because choosing specific types of camera placements etc. already requires semantical knowledge of the scene, e.g. in [43, 121, 140] it is distinguished between higher level concepts for modeling the cinematic expertise and lower level functionalities for designing a cinematographic camera module.

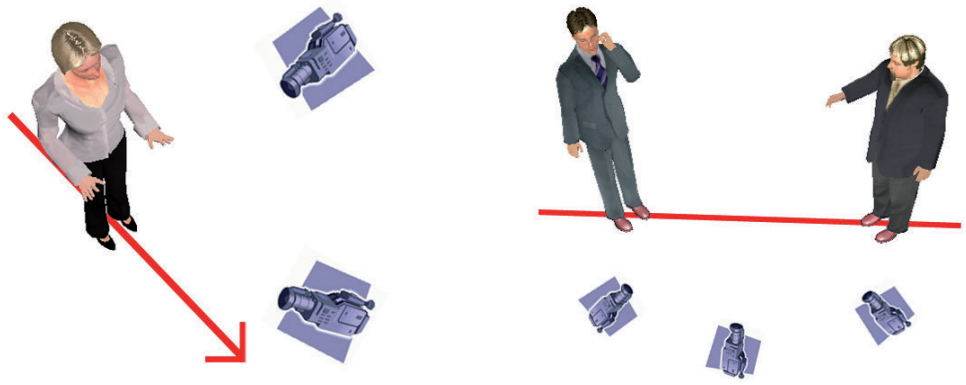

Figure 2.8: *The "line of action" (depicted in red) divides the scene in two parts, which must not be crossed by the camera during a shot – shown for one actor (defined by his facing direction, left), and two actors (given by the line going through both, right).*

2.3.3 Cinematographic Approaches

In the context of film making, Blinn [28] was one of the first to think about where to place a camera to get an interesting picture based on a description of the desired scene (e.g. a spacecraft at (x_s, y_s) and the planet behind). In contrast to the traditional lookAt transformation, which is fully defined by the eye position e in world space, a viewing direction $\vec{v}$, and an additional up-vector, he starts from the given 2D location (x_s, y_s) of the object of interest (e.g. the mentioned spaceship or an actor) in screen space, and then solves for the appropriate camera matrix in 3D world space. More recently, [119] explained how to incorporate filmmaking techniques, including the choice of lenses and filters to use, into games and similar applications.

Thus, special effects like depth of field [277, 114] and motion blur [264] directly result from the camera system used, and can be implemented on modern graphics hardware quite efficiently in a post-processing step. Also, [56] mentioned that not only framing but also the correct choice of lenses, filters and similar effects are crucial for the final perception and can influence a user's perception of emotions.

Already [43] stated, that interactive 3D applications fail to realize important storytelling capabilities and natural interactions with intelligent agents by ignoring the rules of camera placement and thus the principles of cinematography – which are essential for film making but nowadays so common that they are invisible to people. Thus, they formalize cinematographic principles into a declarative high-level language for automatic camera control based on certain idioms (i.e. stereotypes for filming specific actions like a dialog scene). Likewise, [121] organized such idioms with finite state machines on a higher level, whereas several camera modules with different behavior (e.g. 'follow' and 'apex') were responsible for the low-level camera placement.

A similar but knowledge-based approach that is implemented with the help of TVML (cp. section 2.1.2) for visualizing the scene via their TV production and simulation tool for desktop environments is proposed by [120, 140]. In [85], intelligent camera control is utilized for visualizing important events in crowd simulations. Facilitating cinematographic principles (see also section 2.3.2) not only prevents a viewer from becoming disoriented

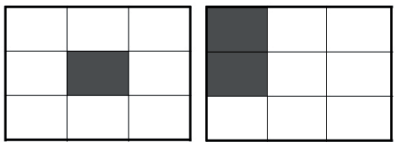

Figure 2.9: *The "rule of thirds" splits the screen into nine parts. The ones marked in gray in both framing symbols [349] exemplarily depict positions where an actor shall appear.*

due to the camera work but it also allows communicating additional information.

As mentioned, one such rule is the so-called "line of action" (see Figure 2.8), which is either formed along the direction a single actor is facing or by the line connecting two interacting actors [119], whereas the camera placement is specified relative to that imaginary line. Furthermore, there are also guidelines concerning the location of an object on the screen, which can be described by the so-called "rule of thirds" (see Figure 2.9) for framing the scene and by utilizing standard shot sizes like "close-up" or "full shot" [43, 182].

Moreover, users often have problems navigating through virtual environments, especially with typical 3D devices. Therefore, [182] propose to apply the rules of filming also to games and other real-time 3D applications. Recently, [171] presented an approach using hierarchical lines of action (see Figure 2.8) for generating correct camera setups for scenes that contain groups of more than two or three actors.

Current tools often used e.g. for pre-visualization in movies like Autodesk Maya require the scene author to go on a very low and technical level. For example, the latest version, Autodesk Maya 2011 [14], promises to accelerate pre-visualization and virtual movie-making production by enabling artists to layout and manage multiple camera shots in a single animation sequence. The timings of shots can be changed and alternate versions can be created and reviewed. But this requires having skilled modelers and animators at hand. There are also some commercial tools available for creating pre-visualization films like FrameForge [134], which is a level-editor-like authoring tool that comes with a comprehensive set of assets and effects. But as it does not provide any high-level interface, it is more comparable to a typical animation engine.

A comprehensive introduction and overview on camera control in interactive applications including cinematographic considerations and high-level declarative approaches is given in [44, 45]. Recently, [150, 341] proposed to adopt cinematographic techniques to X3D. Whereas [150] presented a set of nodes to ease framing dynamic content and including special effects, [341] discussed establishing higher level cinematographic concepts by introducing nodes that encapsulate camera movements and shots and support offline rendering.

As a concluding remark, finally it is worth mentioning that similar to a movie a feeling of immersion also can be achieved by means of a good story, which furthermore serves as a leitmotif and provides the user with background information and structure for helping him to understand the concepts of the application and to build adequate mental models. Last but not least narration leads to emotional engagement and the feeling of presence. Thereby, one day virtual human experiences may be ubiquitous in info- and edutainment, especially in combination with other natural interfaces.

3 Real-time Rendering and Simulation Basics

This chapter consists of two main parts. Firstly, physically-based simulation and lighting in general are discussed, which are relevant for simulating and visualizing specific properties of human physiology such as hair and skin. The latter are outlined secondly.

3.1 Deformable Objects

Especially in the area of deformable objects, e.g. cloth and hair simulations, virtual characters offer various fields of research. Many algorithms have been developed for the offline area and so far were not applicable to real-time applications because of computational effort and insufficient robustness, though this is currently changing due to the proliferation of multi-core systems and freely programmable GPU architectures.

3.1.1 Fundamentals of Physics

In the following some fundamentals necessary for doing physically-based simulations are outlined [92, 202]. Whereas statics considers forces at equilibrium, kinematics deals with the motions of a body, whose causes are studied by dynamics. These areas belong to mechanics. Here an object is idealized as a mass point or a rigid body with mass m, whose form does not change under the influence of external forces.

Basic units for determining rotational and translational motions are velocity v and acceleration a (with way s and time t):

$$v = \lim_{\Delta t \to 0} \frac{\Delta s}{\Delta t} = \frac{ds}{dt} = \dot{s} \qquad a = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{dv}{dt} = \dot{v} \tag{3.1}$$

Thereby, for uniform acceleration (i.e. $a = const$) the following physical law (in vectorial form) results for the way s, with index 0 referring to start time $t = 0$:

$$\vec{s} = \frac{t^2}{2}\vec{a} + t\vec{v_0} + \vec{s_0} \tag{3.2}$$

Motions are caused by forces. This relation is described by Newton's laws of motion:

First law: A body does not change its velocity in magnitude and direction, if no external forces are acting on it.

Second law: The temporal change of momentum $\vec{p} = m \cdot \vec{v}$ is proportional to the acting force, i.e. for accelerating a body a force $\vec{F}$ is necessary, which equals the product of constant mass m and acceleration $\vec{a}$.

$$\vec{F} = m \cdot \vec{a} \tag{3.3}$$

Third law: Whenever an objects A exerts a force onto another object B, then B simultaneously exerts a force on A with the same magnitude into the opposite direction: $\vec{F_A} = -\vec{F_B}$.

Apart from translatory motions rigid bodies can also perform rotary motions. Besides angular velocity $\vec{\omega}$ (with $\vec{v} = \vec{\omega} \times \vec{r}$) and angular acceleration $\vec{\alpha}$ (with $\vec{a} = \vec{\alpha} \times \vec{r}$), therefore the quantities torque $\vec{M}$ and moment of inertia $J = \int r^2 \mathrm{d}m$ are of relevance. Based on the well-known lever rule ($r_1 \cdot F_1 = r_2 \cdot F_2$) torque is defined as $\vec{M} = \vec{r} \times \vec{F}$, where $\vec{r}$ is the position vector from the origin of rotation to the point of force application. Analogously to equation 3.3 the dynamic balance for circular motions is obtained by $\vec{M} = J \cdot \vec{\alpha}$. Here, analog to the momentum $\vec{p}$, the angular momentum $\vec{L}$ is determined by $\vec{L} = J \cdot \vec{\omega}$.

There are various external forces that can act onto a body such as gravity with the gravitational acceleration $g \approx 9.81 \frac{m}{s^2}$ or frictional forces. Although the latter are motion-resistant forces they can transmit power and thereby enable acceleration. Apart from dry friction there also exists fluid friction. Here, within a certain boundary layer contact interaction occurs between the moving surface of a body and a liquid or gaseous medium.

The relationship between spring force F and elongation Δs for elastic deformations is described by Hooke's law of elasticity (equation 3.4), where D is the spring constant that determines the spring stiffness.

$$F = D \cdot \Delta s \tag{3.4}$$

The most important property of deformable objects is their elasticity [70, page 161], i.e. the ability to return to the initial shape after a deformation caused by external forces. In contrast to mechanics, in continuum mechanics materials are modeled as continuous mass instead of discrete particles, and thereby the deformations of solids and fluid under the influence of external forces and torques are considered, too [92].

Instead of examining system properties that affect a body as a whole, like mass m, force $\vec{F}$ and deformation Δs, specific material properties like density $\rho = \frac{m}{V}$, stress tensor σ (with $\vec{F} = \sigma \cdot \vec{a}$), and strain $\epsilon = \frac{\Delta s}{s_0}$ are considered. By introducing another parameter analogous to the spring constant D, namely Young's Modulus E (with $E = \frac{F/A}{\Delta s / s_0}$), for the strain the relationship $\sigma = E \cdot \epsilon$ holds. The same goes for shear strain $\tau = G \cdot \frac{\Delta s}{s_0} = G \cdot \gamma$ with the torsion modulus G.

3.1.2 Flow Behavior

Due to the lack of solid shapes and the effects of various inner forces, fluids exhibit a very complex behavior, which is still not yet fully modeled by the laws of physics. Thus,

the plausible simulation and rendering of fluids in real-time is still a challenging task in computer graphics. Depending on the dimensions of the simulated fluids and their interactions with the surrounding environment (e.g. a big lake, the surge at a seashore, a stream, or rain-drops on a window pane), different behavioral aspects are important.

Nevertheless, most of the previous research in graphics and computational fluid dynamics (CFD) focused on large bodies of liquids like an ocean. Boundaries like a seashore were either avoided or the surface tensions between liquid and solid were ignored. Although this is possible for fluid simulations at a larger scale, in the case of small-scale fluid motions, such as water drops flowing on a glass window as shown in Figure 5.19 (page 159), surface and inter-facial tension effects become extremely strong and must also be handled.

Fluid Dynamics

Flow phenomena are studied in hydrodynamics, a subdiscipline of fluid dynamics [92, 252]. They differ from solids in that their particles are easily movable and only small forces are necessary for a deformation. This behavior is called fluid flow. Basically there exist two types of fluid flows. Laminar flow occurs in slowly flowing fluids and is a spatially and temporally regular stratified type of flow. Turbulent flow in contrast is characterized by faster flows with recirculation and spiral eddies.

The flow behavior is caused by the interaction of molecules and the boundary layer. This causes momentum transfer between fluid molecules as well as from boundary to fluid molecules. This leads to a small region along the solid body, which exhibits a continuous decrease in speed perpendicular to it. The fluid molecules directly on the surface of the solid do no longer exhibit any motion because here the influence of the solid body molecules is at a maximum. Emanating from and perpendicular to the boundary the speed increases from zero to the average flow velocity, which is called the no-slip condition.

Flows with similar geometry can be compared by studying the acting forces. The ratio of inertial to frictional force is called the Reynolds number $Re = (d\nu\rho)/\eta$ [92, page 110], which is dimensionless and characterizes the flow resistance of a viscous fluid. For big values of Re inertial forces prevail and for small ones frictional force prevail.

For every fluid flow there exists a critical Reynolds number Re_k, from which a laminar flow fades to a turbulent flow. This depends on the average flow velocity v, the density ρ, the viscosity η, and the characteristic length d, like e.g. the diameter of the pipe. The so-called kinematic viscosity is denoted by $\nu = \frac{\eta}{\rho}$. Fluid flows that exhibit the same values for Re despite different physical quantities are called mechanical similar.

Though in contrast to solid bodies fluids do not have a constant form, as mentioned for fast deformations a resistance occurs that is called (dynamic) viscosity. But for fluids like water this viscosity is very marginal. In addition, the change in resistance can increase or decrease with higher deformation speed. For fluids with Newtonian behavior the shear strain γ increases linearly with time and G becomes time dependent, which results in the shear stress $\tau = \eta \cdot \frac{\mathrm{d}\gamma}{\mathrm{d}t}$. Shear rate $\mathrm{d}\gamma/\mathrm{d}t$ and shear stress τ are thereby proportional to each others, whereas η is given in $\mathrm{kg}/(\mathrm{m} \cdot \mathrm{s})$.

For non-Newtonian fluids [186] in contrast, like pseudoplastic fluids where the rise in shear stress compared to Newtonian fluids decreases with higher shear rate, there is no direct proportionality. An examples is blood, which for small shear forces exhibits high

viscosity. But for stronger forces the red blood cells are elongated elastically into the flow direction and contracted in their width, which allows blood to flow through small vessels more easily. Shear thickening fluids behave contrariwise. Following [186, p. 40], all fluids approximately can be described like follows: $\tau = K \cdot (\frac{\mathrm{d}\gamma}{\mathrm{d}t})^n, n > 0$, where for $n > 1$ shear thickening fluids and for $n < 1$ pseudoplastic fluids are described.

The rationale behind the conservation of mass is expressed with the continuity equation, which basically states that at gorge portions there are higher flow velocities, i.e. the volumetric flow rate $\Phi = \vec{v} \cdot \vec{a}$ remains constant for a volume element, given that the medium is incompressible, i.e. $\rho \neq \mathrm{f}(p)$, and the system does neither contain sources nor sinks [252, p. 202]. Generally spoken, with flow density $\vec{j} = \rho\vec{v}$ the following physical relationship (equation 3.5) holds. For incompressible fluids, where $\rho = const$ and $\frac{\partial\rho}{\partial t} = 0$, equation 3.5 thereby further simplifies to $\nabla \cdot \vec{v} = 0$, which means that the vector field defined by the fluid's velocity has zero divergence.

$$\frac{\partial\rho}{\partial t} + \nabla \cdot \vec{j} = \frac{\partial\rho}{\partial t} + \frac{\partial(\rho \cdot v_x)}{\partial x} + \frac{\partial(\rho \cdot v_y)}{\partial y} + \frac{\partial(\rho \cdot v_z)}{\partial z} \stackrel{!}{=} 0 \tag{3.5}$$

The most generic form of the motion equations for Newtonian fluids are the Navier-Stokes equations, a system of partial differential equations of second order, which is the main basis in the area of CFD, where the equation system is solved with numeric approximation methods like the finite-difference method. For homogeneous, incompressible fluids such as water thereby the following equation 3.5 of momentum[1] is obtained for a stationary volume element to describe the temporal change of velocity $\frac{\partial\vec{v}}{\partial t}$ [116, 252]. Here, p is the scalar pressure, ν the kinematic viscosity, and $\vec{f}$ are additionally acting volume forces. Since homogeneity is assumed, the fluid density ρ is constant in space, and due to incompressibility the fluid volume is constant over time and thus $\nabla \cdot \vec{v} = 0$.

$$\frac{\partial\vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\,\vec{v} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{v} + \vec{f} \tag{3.6}$$

The surface of a fluid, which especially for the consideration of droplet flow is of interest, tends to diminish itself, because neighboring molecules attract each others due to intermolecular forces, the cohesive forces, which cause molecules on the surface to move into the fluid's interior. Thus only those few molecules remain on the surface, which are inevitable to build the minimum surface area – the so-called minimal surface. Here, surface tension is a force C that is directed tangentially to the surface keeping it in balance.

If the boundary layers of various fluids converge along one edge, certain angles are formed due to the equilibrium of forces of the interfacial tensions C_{12}, C_{13}, and C_{23} [252, p. 37]. In case the surface tension of one fluid is higher than the sum of both others, no equilibrium is formed. And if one of the three materials is solid (as shown in Figure 3.1), an equilibrium between the other two materials is formed, where C_{12} denotes the surface tension between those. The contact angle α then is obtained using trigonometry.

[1] $\nu\nabla^2\vec{v}$ are frictional forces and $(\vec{v} \cdot \nabla)\vec{v} = \nabla\frac{v^2}{2} - \vec{v} \times (\nabla \times \vec{v})$ is the convective acceleration.

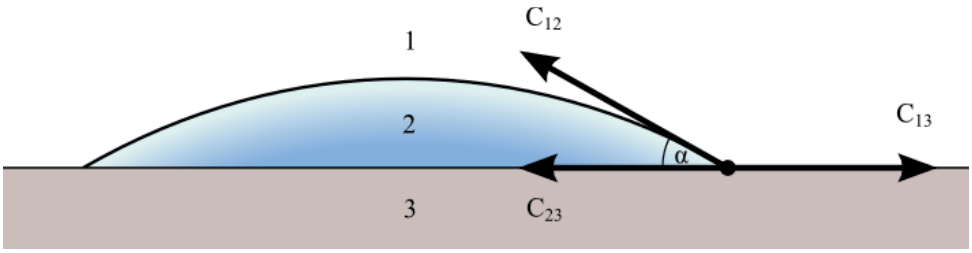

Figure 3.1: *Contact angle α between two fluids (1, 2) on a solid surface (3).*

Droplet Flow

On-surface fluid flow is not only of interest in the area of natural phenomena in general, but also for simulating more life-like virtual characters, because it allows modeling a dynamical and plausible flowing of blood, sweat, tears and so on (the tears in Figure 5.13 are colored exaggerated to enhance contrast). Furthermore, effects like rain and tears can convey or alter certain kinds of emotions, which is important for computer games etc.

For on-surface droplet flow, the appearance and flow behavior is different from typical fluid dynamics and mainly influenced by factors such as surface adhesion or the formation of minimal surfaces. As will be outlined next, for interactive applications the animation of droplet flows can be mainly based on phenomenological observations. Thus, already Kaneda et al. [169] mention the following factors as being most relevant here:

Flow condition: A drop begins to flow down a slope, if it exceeds a critical mass, based on the amount and viscosity of the fluid as well as the angle of inclination of the surface.

Meandering flow: When flowing down, the drop makes meandering motions, which seem to be random, but are caused by taking the line of the least resistance.

Water trails: Drops leave a trail of liquid consisting of smaller drops behind, and because of the fluid loss, the drops finally grind to a halt. The resulting path then impacts the behavior of other drops, because here the resistance is less than at dry places.

Merging drops: Confluent drops merge to one drop, whose velocity depends on the velocity and mass of the original drops.

Based on the factors mentioned above, Kaneda et al. developed an empirical model for simulating droplet flow [169, 170, 168], by covering a surface with a discrete 2D grid. In every grid cell, drops can originate, jump to the next grid cell, and merge with other drops. Their method is sketched in Figure 3.2. First, the drops are placed on the grid (maximal one water droplet per cell), and then, drops and grid elements are initialized with their physical properties (mass, affinity, and velocity). The water affinity denotes the surface roughness and is used for meandering. After that, the most probable next flow direction d, based on Newton's second law of motion, the given water affinity, and the already present wetness, is evaluated stochastically.

Originally, this model was designed for offline rendering and took several seconds for one frame, but it is nowadays also suitable for real-time applications. Based on Kaneda's model, in [307] a GPU-based implementation for simulating rain-drops on window panes was presented. A related method for animating the flow of water droplets on structured but flat surfaces was proposed in [142]. Here, the drops are represented by 3D particles

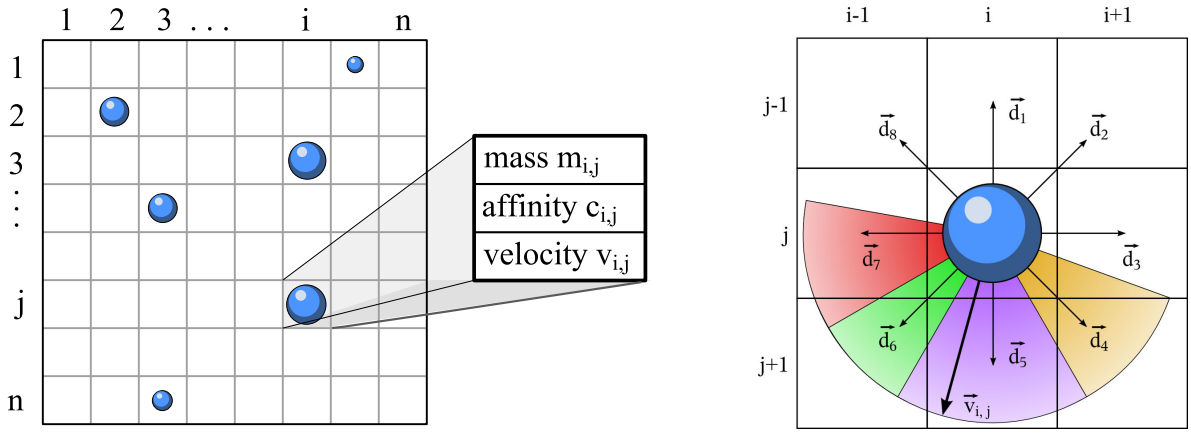

Figure 3.2: *Physical properties of water droplet for every grid element (left) as well as possible directions d_k to neighboring grid elements and velocity vector $v_{i,j}$ (right), after [168].*

and affected by the underlying bump mapped surface. Instead of real geometry, for speed-up the bump normals are used to control the motion of the droplets. A real-time method for animating droplet flow in the case of tears and sweat based on pre-defined key-frames encoded in a 3D texture was presented in [155].

Recently, another algorithm to simulate tears based on Smoothed Particle Hydrodynamics, which acts on a 3D grid and therefore is not restricted to image space, was proposed by van Tol and Egges [315, 316]. The simulated fluid then is visualized by generating a mesh each time step, using the marching cubes algorithm [205] for determining the resulting isosurface. But as opposed to image-space-based approaches (e.g. Tatarchuk [307]), it is computationally more complex and does not scale well with the number of droplets.

An approach that was directly intended for GPU implementation, recently was proposed by El Hajjar et al. [112]. Similar to Kaneda et al. [168], the authors also use a grid for describing the fluid distribution on the surface. But here, a grid element does not denote a single drop (as shown in Figure 3.2), but a much smaller amount of fluid that, together with neighboring elements, makes up a complete drop. Droplets are placed by rendering a disk, with additive blending enabled, into the liquid texture, which keeps the velocity $\vec{v}$ and fluid volume q for all grid elements in its color channels.

The velocity $\vec{v}$ is updated for every time step Δt by calculating the new acceleration $\vec{a}$ in tangent space by considering gravity and friction. Because in shader programs only gather but no scatter operations are possible (see section 3.2.3 for some details), the fluid transport for a given grid cell then is computed implicitly by calculating the amount of fluid that will flow into that cell from its eight neighbor elements.

A physically-based method for simulating drops is presented in [329], which achieves very realistic results and in contrast to other approaches also can simulate fluids dripping off a surface. This is realized by introducing the so-called virtual surface method for explicitly modeling fluid-solid interactions, but it requires several days for computation and is thus not suitable for interactive applications.

In [203], nonzero divergence (i.e. $\nabla \cdot \vec{v} \neq 0$) in the 2D Navier-Stokes equations is considered for simulating water absorption from on-surface flows, by also incorporating erosion and

deposition, but surface tension effects are not reflected by the resulting droplet shapes. Harris [116] explains, how to solve the Navier-Stokes equations for incompressible, homogeneous flows (i.e. $\rho = const$ and $\nabla \cdot \vec{v} = 0$) for the 2D case in general based on GPGPU techniques (see equation 3.6), though leaving boundary conditions mostly aside.

3.1.3 Deformation Methods

This section briefly touches on methods to simulate deformable objects. In computer science there exist different methods to simulate deformations. Usually one distinguishes between physically-based deformation methods, like the finite element method (FEM), and geometrical ones. The latter only change the shape of polygonal models and are suitable as a tool for intuitive deformation tasks such as in modeling- or CAD-packages.

A well-known geometrical method is the grid-based free-form deformation (FFD) [280], where the geometries to be deformed are embedded into a 3D lattice whose grid points serve as control points for a Bernstein polynomial used for deformation. However, other deformation schemes are possible, too, such as the mass-spring-based simulation presented in [322], which combines this indirect deformation method with a more physically-based type of mesh deformation to animate complex hairstyles.

To achieve an animated deformation using physically-based methods, e.g. of cloth or hair, a simulation system needs to be advanced in time. Therefore, the motion equation needs to be solved for the given time step, which usually is carried out with numerical integration of a differential equation. The type of differential equation to be solved depends on the type of motion equation. Newtonian motion is based on Newton's second law of motion (section 3.1.1) and can be solved by integrating an ordinary differential equation (ODE). The implementation is relatively easy because of the discretization to a finite number of mass points, and the integration is independent for each such particle.

For continuous models as required in continuum mechanics Lagrangian motions are used, which consider the path in a vector field instead of discrete particles and locations. In contrast to particle-system-based models a continuum mechanics model allows for a much more precise simulation of continuous solids and fluids, but since this requires time intense computations, we only consider classical mechanics for the simulation methods presented in this thesis by using a grid-based Eulerian frame of reference.

Generally, a differential equation is an equation that relates an unknown function f to its derivatives of various orders, in which it is distinguished between ordinary and partial differential equations (PDE). In contrast to the latter, an ODE only has one independent variable, for instance some x or the time t Since differential equations often can't be solved analytically, the solution usually is computed with numerical integration.

Based on the initial value problem $y' = \mathrm{f}(t, y)$ with the initial condition $y(t_0) = y_0$ (visualized to the right in Figure 3.3) and the relationship $\mathrm{y}' = \lim_{\Delta t \to 0} \frac{\mathrm{y}(t+\Delta t) - \mathrm{y}(t)}{\Delta t}$, this yields, by approximating that term with the tangent (with step size Δt) and rearranging it, the following differential equation that is also known as the explicit Euler scheme:

$$y_{n+1} = y_n + \Delta t \cdot \mathrm{f}(t_n, y_n) \tag{3.7}$$

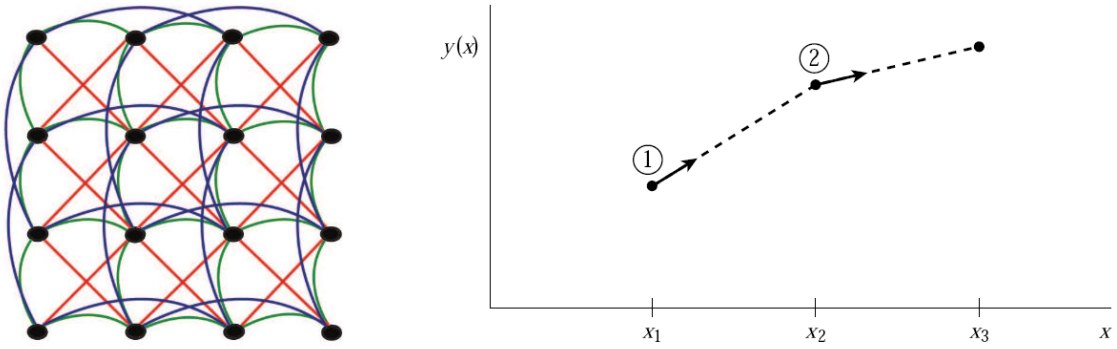

Figure 3.3: *Left: different spring types in a mass-spring-system. Right: Visualization of Euler method for numerically calculating an initial value problem (cf. [255, p. 711]).*

Equation 3.7 is a frequently used method for mass-spring-system-based simulations and the like. Figure 3.3 (left) shows an example of the underlying structure, where the objects to be deformed (typically fabrics or hair) are discretized by several mass points that are connected with certain spring types. The differential equation to be evaluated basically follows from identifying Newton's second law of motion with Hook's law. An extensive discussion including implementation hints for instance can be found in [70, 346].

Under the assumption of differentiability, equation 3.7 mathematically corresponds to a Taylor series expansion, which is truncated after the linear term. With an error term of order $\mathrm{O}(\Delta t^2)$, a method error of order $\mathrm{O}(\Delta t)$ results [255, 346], as is apparent from the following series representation:

$$\mathrm{y}(t) = \sum_{k=0}^{\infty} \frac{\mathrm{y}^{(k)}(t_0)}{k!} \cdot (t - t_0)^k \;\Rightarrow\; \mathrm{y}(t_0 + \Delta t) = \mathrm{y}(t_0) + \Delta t \cdot \mathrm{y}'(t_0) + \frac{\Delta t^2}{2!} \cdot \mathrm{y}''(t_0) + \ldots$$

Besides its inaccuracy, which increases with bigger step sizes, the main problem of Euler's method is its instability. If the step size Δt is too big, the numeric solution does not converge but oscillates or the system even blows up, which gets clear when remembering that only the tangential gradient is considered. This issue is also known as stiffness [70], which for example in a mass-spring-system denotes the ratio between spring constant D and time step Δt (compare equation 3.4).

Here the system is called stiff, if for a given spring constant D the time step needs to be chosen unacceptably small for obtaining a stable solution [70, 255, 346]. In this case improvements like Verlet integration and second or fourth order Runge-Kutta are more efficient, since t_n and t_{n+1} are considered at the same time and thereby the previously mentioned error term is minimized through a better approximation of the function.

If in equation 3.7 one moves on from y'_n to y'_{n+1}, this is called implicit integration, which is numerically more stable and nearly independent of the step size. In contrast to explicit methods, for implicit methods the approximated derivative of the subsequent state is used as the integration basis instead of the current derivative. However, implicit methods require solving a linear equation system for each time step [18, 346].

3.2 Real-time Rendering

For being displayed on the screen, the 3D objects need to be rendered, whereas the input data is used to control the 3D API. Besides offline global illumination (GI) approaches like raytracing [285] and radiosity [287], in real-time rendering local lighting models in combination with rasterization via dedicated graphics hardware is prevalent. While some years ago the CPU was responsible for all rendering tasks, nowadays most 3D calculations are taken over by specialized graphics hardware, the GPU (Graphics Processing Unit).

In this regard, two different 3D APIs (whose implementation in general is part of the graphics driver) are common: Direct3D, which is optimized for game development and part of Microsoft's DirectX and thus only available on Windows platforms, as well as OpenGL [174], which was designed as a state machine and is a platform independent standard that is controlled by the OpenGL Architecture Review Board (ARB).

The core component of real-time computer graphics is the rendering pipeline, which consists of three main stages, the application, geometry, and rasterizer stage [2]. As can be seen in Figure 3.4, the application stage is executed on the CPU. This stage is responsible for instance for collision detection and frustum culling, and explained in more detail in the first subsection. The geometry stage make up the lower left part in Figure 3.4 and is responsible for model and view transform, vertex shading, projection, clipping, and mapping to 2D window coordinates.

The rasterizer stage (lower right part in Figure 3.4) finally is responsible for triangle setup, scan conversion, pixel shading (e.g. texturing), and per-fragment operations (e.g. depth test and alpha blending). Typically, both latter stages are hardware accelerated by executing them on the GPU. Whereas some years ago with the first GPUs most functionality was hard-wired by the so-called fixed function pipeline, nowadays programmable shaders allow developers not only to implement their own shading algorithms but also to do general-purpose computations on the GPU (GPGPU).

3.2.1 Application Models and X3D

An established application model is the scene-graph [2, 109], which is a very common and flexible data structure for 3D applications to organize the data as well as spatial and logical relations, thereby facilitating operations like culling. From a graph theory point of view, it is a directed acyclic graph (DAG) that contains and hierarchically organizes all scene objects in its nodes. A minimal scene-graph consists of three node types: groups, transformations, and geometries, whereas the latter can also provide information about the material and usually are leave nodes. The root node thereby contains the whole scene, whose children represent scene objects, which in turn can be roots of a sub-hierarchy.

Attribute inheritance happens from the parents to the children. Therefore, the scene objects do not need to be transformed individually, but automatically are transformed according to the parent nodes' transformations. Because each subtree has its own locale coordinate system, during hierarchy traversal the individual transformations are accumulated to determine the global world space position. Thus, scene-graphs are specially suited

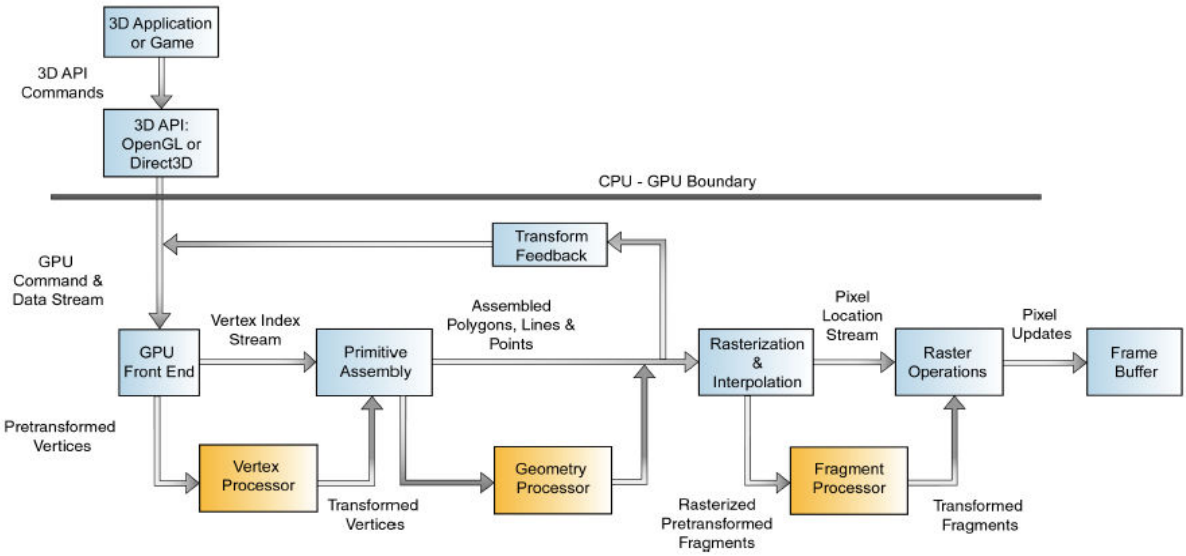

Figure 3.4: *The rendering pipeline – programmable stages are marked in amber (cf. [117]).*

to render complex scenes efficiently in that for instance only potentially visible elements are rendered by culling all objects outside the view frustum.

Hence, for scene management most developers of Virtual Reality (VR) and similar 3D applications utilize scene-graph libraries like OpenSG [258] or Open Scene Graph [231], which all come with a proprietary scene and file structure. Other well-known examples of scene-graph systems are Java3D [301], Open Inventor [297], Performer, and NVIDIA's more recent SceniX scene management engine [228], a successor of the NVIDIA Scene Graph that supports all modern GPU features.

Besides older monolithic systems there exist VR frameworks that support and provide even higher abstraction for behavior and animations like Avocado [312] or VHD++ [251], but they are not based on standards. Therefore, with Avalon a scalable framework for dynamic MR applications was presented in [22], which utilizes the VRML/ X3D standard [336] as basis for an application programming and description language, since X3D defines features like the scene-graph that are also useful for VR/ AR environments.

The semantics of the X3D ISO standard describe an abstract functional behavior of time-based, interactive 3D, multimedia information. It is independent of any specific software or hardware setup. However, X3D clients and applications today are mainly built for desktop systems running a web browser. The X3D standard includes all usual node types to describe transform, geometry and material objects. Additionally X3D provides the ability to model the event flow and application behavior by connecting input/ output slots of nodes. These connections are called ROUTEs and they link nodes, which are also part of the scene-graph, to a so-called behavior-graph. In contrast to this, other systems define different nodes types for different edge types (e.g. hierarchy and event-flow).

Moreover, the X3D specification includes high-level device-independent sensor nodes for describing interactive parts of the scene. X3D also combines the scene- and behavior-graph with a powerful scripting interface, which provides a very flexible runtime environment. Application development is done by instantiating, scripting and connecting nodes.

These networks of nodes are stored in a number of files that are portable between different run-time environments. Thereto, the X3D standard supports three encodings, an VRML-compatible classic encoding based on Open Inventor [297], an XML-based modern encoding, and a binary-compressed encoding to minimize size and loading time.

One major drawback of X3D is the extremely limited support for IO devices. The X3D specification does not mention devices at all – it only specifies some very high-level nodes that allow controlling the way the user navigates in the scene and interacts with objects. The actual mapping between these nodes and the concrete devices connected to the computer is up to the player. While this interaction model is sufficient for web-based 3D applications consisting of simple walk-through scenarios, it is much too limited for immersive VR and AR applications. For example, consider a video see-through AR application where the X3D scene needs to get the video images from a camera attached to the system to put them into the background of the virtual scene.

To alleviate these drawbacks, Behr et al. [22, 23, 24] propose a layered approach to integrate support for IO devices into X3D, which is integrated into the Instant Reality framework [135]. On the basic layer a set of low-level sensors is proposed that allow receiving data streams from or sending data streams to devices or external components. On top of this layer a set of high-level sensors consisting of the traditional pointing device sensor nodes mentioned in the X3D specification is situated. This approach is similar to that of traditional 2D user interfaces where a low-level layer handles simple mouse and keyboard events and higher-level layers consist of UI elements like buttons and menus.

Their framework also provides two types of network interfaces to incorporate application data at runtime, e.g. from simulator packages or external devices. Application services are external interfaces that provide access to scene-graph elements, and system services are internal interfaces to distribute tasks [23, 24]. The X3D standard already includes the scene access interface (SAI) [337], which can be used to interact with X3D worlds both from within the worlds or from external applications, e.g. using JavaScript or Java. The system services are streaming protocols that are used to cluster parts of the media such as audio. In contrast to the SAI, this layer does not have any knowledge about the scene-graph. One example is the clustering of graphics, which is essential for multi-screen support but also for balancing the rendering load.

3.2.2 Lighting Models

As mentioned, in computer graphics one distinguishes between local and global lighting models. For the first model type only the object to be lit is considered (in most cases the vertex or pixel to be displayed as well as some few light sources) by disregarding the interactions with surrounding objects. The second type, like raytracing, also considers light emanating from or being absorbed by other objects. The light distribution in such scenarios is highly complex and needs to be integrated at all points for all incoming direct and indirect illumination, which is a computationally very intense offline process.

For shading a surface, the outgoing radiance L_o in a given direction is evaluated in terms of the incoming irradiance L_i. The function $f_r(x, \vec{\omega_i}, \vec{\omega_o})$ that describes how a surface reflects light is called BRDF (bidirectional reflectance distribution function [2]). Depending on

this function, the rendering equation [164] models light scattering off various types of surfaces, where Ω is the local hemisphere at x defined by the surface normal $\vec{n}$, $L_i(x, \omega_i)$ is the incident light from ω_i at x, and L_e is the emitted radiance.

$$L_o(x, \vec{\omega_o}) = L_e(x, \vec{\omega_o}) + \int_\Omega f_r(x, \vec{\omega_i}, \vec{\omega_o}) L_i(x, \vec{\omega_i})(\vec{\omega_i} \cdot \vec{n}) \mathrm{d}\vec{\omega_i} \tag{3.8}$$

The BRDF is defined as the ratio between the differential outgoing radiance and incoming irradiance, and can be seen as the relative amount of energy reflected in the outgoing direction for a given incoming direction. In the case of real-time rendering, where only a finite number of non-area light sources are used, this simplifies to the following term, where $\vec{v}$ is the view direction, $\vec{l}$ is the light direction, and $\vec{n}$ is the surface normal:

$$f_r(\vec{v}, \vec{l}) = \frac{L_o(\vec{v})}{(\vec{l} \cdot \vec{n}) \cdot L_i(\vec{l})} \tag{3.9}$$

Obviously, the simplest type of a BRDF is a constant, i.e. $\forall \vec{v}, \vec{l} : f_r(\vec{v}, \vec{l}) = k_d$ (with $k_d = const$), also known as Lambertian BRDF [2], whose reflectance lobe is a simple hemisphere. For being physically valid, it is furthermore required that the BRDF is energy conserving, i.e. $k_d/\pi \in [0, 1]$, and follows the Helmholtz reciprocity [287, ch. 2]. Here, the diffuse term $k_d(x)$ models the perfect diffuse response of a Lambertian surface.

$$\begin{aligned} L_o(x) &= \int_\Omega k_d(x) L_i(x, \vec{\omega_i})(\vec{\omega_i} \cdot \vec{n}) \mathrm{d}\vec{\omega_i} \\ &= k_d(x) \int_\Omega L_i(x, \vec{\omega_i}) \cos\theta_i \mathrm{d}\vec{\omega_i} \end{aligned} \tag{3.10}$$

For interactive applications complex lighting distributions and material properties are reduced to simpler terms that can be evaluated in real-time. Hence, rendering APIs like OpenGL [174] only support a limited number of lights and light types as well as fairly simple material models such as the Lambertian BRDF [86]. Furthermore, the lights typically present in real-time rendering (i.e. directional, point and spot light) have a Dirac delta function in their radiance distribution, so the integral above will transform into a sum over the light sources as shown in the following equation 3.11.

$$L_o(x) = k_d(x) \sum_j L_i^j(x, \vec{\omega_i}) \cos\theta_i{}^j \tag{3.11}$$

In contrast to the BRDF, which assumes that light enters and leaves at the same point x, the BSSRDF f_s (bidirectional surface scattering reflectance distribution function [2]) also takes incoming and outgoing locations into account and thereby allows describing subsurface scattering. As can be seen in equation 3.12, the outgoing radiance $L_o(x_o, \vec{\omega_o})$

is calculated by integrating over all incoming directions $\vec{\omega}$ and area $A(x)$, i.e. over all incoming locations x_i. This is especially important for translucent materials such as skin that exhibit significant light transport below the surface [115, 138].

$$L_o(x_o, \vec{\omega_o}) = \int_A \int_\Omega f_s(x_i, \vec{\omega_i}; x_o, \vec{\omega_o}) L_i(x_i, \vec{\omega_i})(\vec{\omega_i} \cdot \vec{n}) \mathrm{d}\vec{\omega_i} \mathrm{d}A(x_i) \tag{3.12}$$

Bump mapping [30] describes perturbations of the shading normal via normal or height textures to produce a richer appearance. Because usually bump mapping is carried out in tangent space, besides the normals also tangents and binormals are required. They can be calculated automatically based on the texture coordinates via $(\frac{d_i}{d_s}, \frac{d_i}{d_t}, \text{scalingFactor}) := (1, 0, -\frac{d_i}{d_s}) \times (0, 1, -\frac{d_i}{d_t})$, supposed they are mostly continuous and one-to-one. Determining the tangents this way is possible, because texture and tangent space both belong to the class of surface local spaces, which is defined for every point of a surface. In tangent space this point defines the local origin, the z-axis is directed along the geometric normal in object space, and the x- and y-axis are located in the tangent plane [83].

A bidirectional texturing function (BTF) allows using spatial characteristics of certain materials, including complex self-shadowing and reflectance, for realistic shading. Therefore, the data is encoded into n-dimensional textures and suitably mapped onto the geometry. However, memory footprint and processing time are rather high. In [172] hence a GPU-based real-time approximation for BTFs was proposed, where for speed-up the material is assumed to be isotropic (i.e. when rotating around the normal, the reflection properties do not change). Then the BTF can be represented as 3D texture, and the z-index corresponds to the viewing angle θ that is taken as look-up index.

Very recently [218] proposed an editable BTF representation by defining light interaction maps and geometry maps for representing the meso-scale geometry by a depth map. Furthermore, a set of basis materials represented by a BRDF are used to describe the micro-structure, thereby also requiring less texture memory.

A good overview on global illumination rendering methods on the GPU in general can be found in [305]. Precomputed radiance transfer (PRT) is a technique that transforms the rendering equation to frequency space and exploits the rigidness of objects to speed up the rendering process. Light transfer is precomputed and stored in a compact representation inside coefficient vectors or matrices, and does not need to be reevaluated at runtime if the surrounding mesh stays fixed. The integral in the rendering equation can be solved via a dot product of light and transfer vectors within a shader program on the GPU.

In theory, PRT shortens the general rendering equation to an integration of incident lighting and a so-called transfer function T, which contains all other parts of the equation combined together, but ignoring the emitting radiance L_e. Equation 3.8 can therefore be transformed to the first part of equation 3.13. This new equation allows the transformation of its two parts L_i and T to frequency space, exploiting that the convolution integral can be approximated by a dot product of the n first coefficients L_j and T_j.

$$L_o(x, \vec{\omega_o}) = \int_\Omega L_i(x, \vec{\omega_i}) T(x, \vec{\omega_i}, \vec{\omega_o}) \mathrm{d}\vec{\omega_i} \approx \sum_j L_j \cdot T_j \tag{3.13}$$

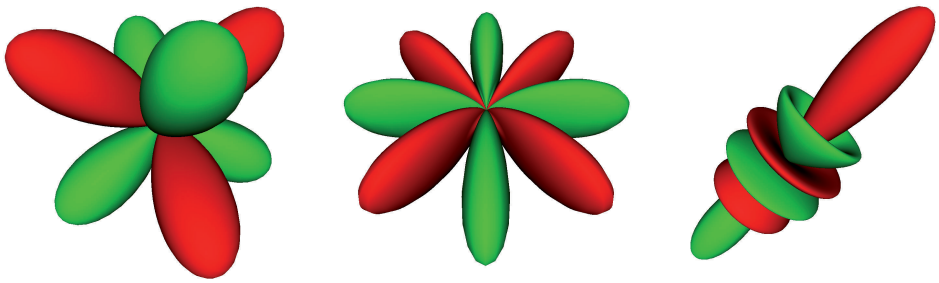

Figure 3.5: *3D visualization of SH base functions y_3^{-2}, y_4^4 and y_5^0.*

Because for rigid objects the transfer coefficients can be precomputed, a band-limited approximation of the rendering equation is made available for real-time rendering. A popular basis function for PRT-like implementations are the real spherical harmonics (SH) functions $y_l^m(\theta, \phi)$, which are conveniently defined in the domain of a unit sphere and can be easily implemented by using a recursive relation [105], given by equation 3.14.

Similar to the Fourier transform spherical harmonics can be used for convolution in the directional domain. This is very useful for integrating the incoming light $L_i(x, \vec{\omega_i})$ over the hemisphere Ω (see rendering equations 3.8 and 3.13), especially in the case of diffuse surfaces and light at an infinite distance. Because for diffuse surfaces the BRDF $f_r(x, \vec{\omega_i}, \vec{\omega_o})$ in equation 3.8 simplifies to a constant term k_d, the equation can be simplified to equation 3.10 with an integral over two directional functions, namely the irradiance L_i and the Lambert term $\vec{\omega_i} \cdot \vec{n}$.

Thereto the – thereby band-limited – transfer function is approximated by a finite function series, and the convolution is replaced by the dot product of the coefficient vectors (equation 3.13). Hence, a low-frequency representation of sphere maps can be efficiently calculated with SH analysis [288]. The base function for these representations is the Legendre polynomial, which can be mapped onto a sphere and redefined for real numbers as shown in equation 3.14. Figure 3.5 shows a visualization of three such functions.

$$y_l^m(\theta, \phi) = \begin{cases} \sqrt{2} K_l^m \cos(m\phi) P_l^m(\cos\theta), & m > 0 \\ \sqrt{2} K_l^m \sin(-m\phi) P_l^{-m}(\cos\theta), & m < 0 \\ K_l^0 P_l^0(\cos\theta), & m = 0 \end{cases} \tag{3.14}$$

$$K_l^m = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} \tag{3.15}$$

One way to precompute the coefficients, in the case that also visibility informations shall be considered, is to use a ray tracer and sample the hemisphere of each vertex in an external tool or during initialization. For speed-up for example the bounding interval hierarchy (BIH) approach proposed in [325] can be used. An important property of spherical harmonics is rotational invariance [105], which allows the construction of a linear transformation $\mathbf{R}$ to project the coefficient vector into a new base without re-evaluating it. A method to construct such a rotation matrix is described in [199].

3.2.3 Programmable Graphics Hardware

Around ten years ago, the first programmable graphics chips were available, which at first allowed programming the vertex and later also the fragment processor. Here, the vertex processor replaces the former fixed-function transform and lighting stage, and the fragment processor replaces the texturing stage, whereas both operate with floating-point precision [84]. Since image and 3D computations involve lots of matrix and vector operations that can be executed in parallel by exploiting data level parallelism, the GPU belongs to the SIMD (single instruction, multiple data) class of computer architectures.

Later, with DirectX 10 or rather Shader Model 4.0, the geometry processor was also added, as shown in Figure 3.4. A characteristic of this new chip generation is the "unified shader" concept, which allows a flexible allocation of the processing resources to the individual stages instead of dedicated processors [117]. Very recently Shader Model 5.0/ OpenGL 4.1 (or DirectX 11 respectively) were introduced with the latest graphics board generation such as NVidia's Fermi chip.[2] This development comes along with the introduction of two other programmable units within the OpenGL pipeline, namely the tessellation control and the tessellation evaluation processor, which are located between vertex and geometry processor and allow subdividing polygonal primitives on the GPU.

Shader Programs in X3D

Specifying surface appearance in a way that balances expressiveness versus portability is a long standing problem in graphics. Similarly to other programming languages material specifications can be classified into imperative and declarative approaches. Imperative approaches specify material behavior with shader programs, often written in a domain-specific language and thereby tied to a specific rendering pipeline. Declarative approaches provide greater abstraction and thus enhance portability and ease of specification, but authors may not be able to specify the appearance as exactly as with imperative approaches.

X3D [336] offers both, appearance properties like the "Material" node declarative option, and the Shaders component as imperative option. The introduction of real-time programmable shading in general into X3D including some design considerations was discussed in [55]. Furthermore, the authors also addressed the obvious separation of the X3D shaders component from the declarative nature of X3D by proposing to establish a prototype library of common shading effects as well as to use appropriate Digital Content Creation (DCC) tools for shader development.

Moreover, it was already outlined how multi-pass effects that require multiple rendering stages could be integrated into X3D. A first but rather non-intuitive proposal for providing access to low-level rendering modes in X3D can be found in [26]. However, multi-pass rendering techniques [65] never made their way into the X3D standard, mainly due to concerns regarding sorting issues and the like.

In X3D, GLSL shaders (OpenGL Shading Language, [266]) are written within a "ShaderPart" node. The shader code can be either embedded within the X3D file or in another file containing only the shader program. There exist two types of "ShaderPart" nodes, vertex and fragment, which we have extended by a third type [135], namely geometry, for

[2] See http://developer.nvidia.com/object/opengl_driver.html and http://www.opengl.org/registry/

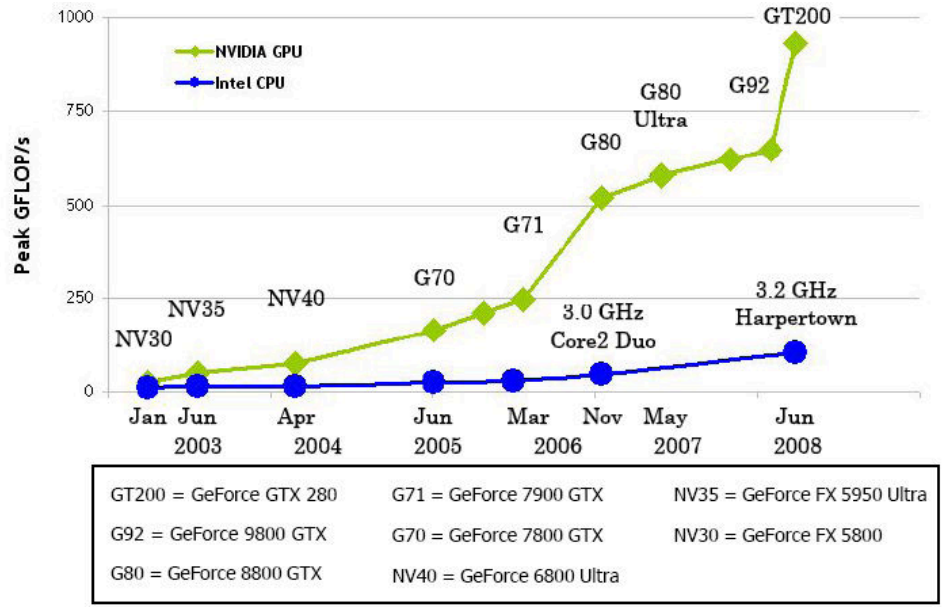

Figure 3.6: *Comparison of performance development of CPU (in blue) and GPU (green).*[3]

additionally supporting geometry shaders, which were introduced with Shader Model 4.0/ Direct3D 10.0 hardware (e.g. NVidia GeForce 8800 graphics card). The program stages for these three shader types are visualized in Figure 3.4.

GLSL uniform parameters can be set via the dynamic field mechanism: for each uniform variable needed, a field with the same name and data type can be defined in the "ComposedShader" node. In order to define textures (uniform variables of any sampler type), an exposedField of type SFInt32 referring to the appropriate texture unit (e.g. 0 for the first one) has to be defined (texture access for vertex shaders needs at least Shader Model 3.0 hardware). Other shader languages like HLSL and CG [83] can be used similarly.

In the vertex shader the incoming vertex position is transformed with the current modelview projection matrix. After that, all varying parameters, which are required by the subsequent fragment shader and interpolated across the primitive during rasterization (like texture coordinates, normals, eye and light vector), are calculated. The fragment shader, which calculates the final fragment color and is responsible for texture access etc., usually realizes a specific lighting model such as Blinn-Phong [2, 86]. Another application domain for shader programming is imaging.

GPGPU

Originally graphics boards were specially designed chips for accelerating computer graphics, whereas today's GPUs are general-purpose high-performance parallel processors that are capable of high computation and data throughput and also support generic programming interfaces such as CUDA and OpenCL [102, 227]. Figure 3.6 shows a comparison of the performance developments of CPU and GPU over the last years. As can be seen, porting an application to the GPU often achieve speed-ups of orders of magnitude versus CPU-based implementations.

[3] Taken from http://www.hardwareinsight.com/wp-content/uploads/2009/10/gpu_vs_cpu.PNG

Following its SIMD architecture, for GPGPU applications, which are mostly applications that require massive vector operations on data of similar type, the GPU acts as a stream processor. Before the advent of modern GPUs with generic APIs like CUDA, or if simply context switches shall be avoided, the corresponding program has to be disguised as graphics application. Therefore, all calculations within a loop have to be identified as kernel that is executed on the incoming fragment [102, 235].

After loading the kernel (i.e. activating the shader program that implements the required operations) and defining the input and output arrays (i.e. input texture and render target), the data stream is generated with help of a *draw()* call (e.g. by rendering a window-sized view-aligned quad whose material consists of the shader program and the input texture), which usually is bound to another texture target as output [102].

Due to constraints of the underlying hardware, concurrent read/ write operations in the same memory area (i.e. using the bound texture as render target) are not possible. Furthermore, a shader only allows gather but no scatter operations. For one thing, gather here means that textures can be accessed at random (though this might be expensive for lots of look-ups). And for another, the nonexistence of scatter operations implies that a shader can only write to the fragment's pixel position, but it cannot change it.

3.2.4 Real-time Shadows

Real-time shadows are needed for depth cues and correct perception of a 3D scene. They can be computed by using e.g. plane projections (a simple technique that only works for planar surfaces), shadow mapping, and shadow volumes. Although the latter method yields very accurate shadow edges, besides being heavily multi-pass-based, it additionally requires the creation of shadow volume geometries for every object that shall throw shadows based on its silhouette edges [48], and therefore scales bad with size and complexity of a scene. For arbitrary scenes current shadow mapping algorithms are much better suited, because they are fast and nearly independent of the scene complexity. Hence, in this section we will mainly review shadow mapping techniques. An introduction and comprehensive overview on real-time shadows can also be found in [2, p. 331 ff.].

Shadow mapping [344] is a multi-pass rendering method [65] based on the idea of first rendering the scene from light view (Figure 3.7, left). Whereas for directional lights a suitable 3D position outside the scene must be calculated, for point lights analogously to a cube map six main directions need to be considered. To find out, if a fragment lies in shadow, it must be tested if there is any occluder between this fragment and the light source. Thereto, first a depth map is created containing the depth information, which in a second pass is projected onto the scene via projective texturing. By transforming each point into light space and comparing the resulting z-values with those from the map, it can be decided if the point lies in the shadow or not. This happens, if the distance to the light is greater than the corresponding value in the depth map.

There exist various shadow mapping techniques. First of all, we have hard shadows like standard shadow mapping, which is very easy to implement and even runs on old graphics hardware, but suffers from two main problems [124]. For one thing they suffer from aliasing artifacts due to the limited shadow map resolution, because different points of

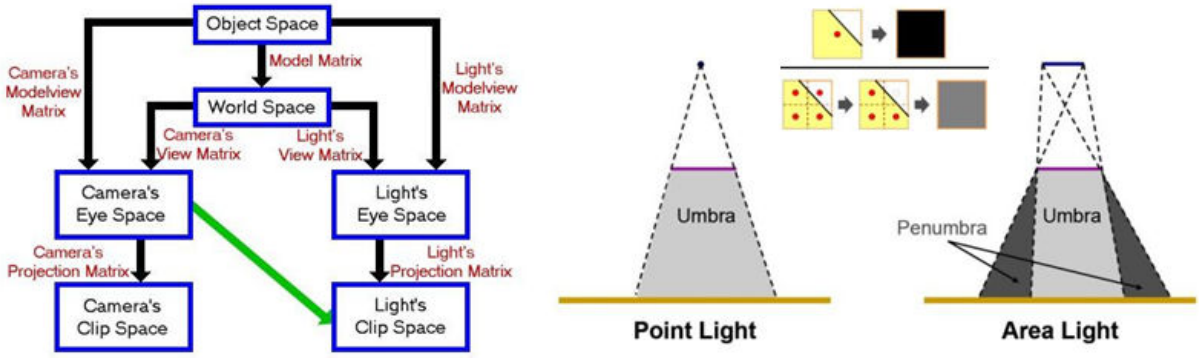

Figure 3.7: *Left: for generating the shadow map and projecting it onto the scene geometry several coordinate transformations are necessary. Right: umbra vs. penumbra (lower part) and principle of PCF filter (upper part), cf. [81].*

the scene are mapped to the same pixel in the shadow map. And for another, similar to z-fighting the depth comparison can fail for points that lie closely together due to the finite z-buffer precision, which gets worse with increasing distance to the light source. Therefore, the light camera's near and far clipping plane needs to be adjusted appropriately and moreover, a suitable bias for the polygon offset for shadow comparison is necessary, especially for big scenes with small objects.

To alleviate these issues, various improvements of shadow mapping algorithms were proposed, because naïvely increasing the shadow map size is only possible to a certain extent, as the maximum texture map size is limited and this also means a considerable loss in performance. On the one hand we have high quality hard shadows like perspective shadow maps [295], which are generated after perspective transformation in the normalized device coordinate space to obtain a higher resolution near the camera by simultaneously minimizing unused areas in the shadow map, as well as the light space perspective shadow maps (LISP) [345], which introduce certain improvements. Both methods have the advantage that the graphics board only needs to support the *shadow* and *depth_texture* OpenGL extensions, which are part of the core since OpenGL 1.4.

For perspective shadow mapping first a so-called "body" is generated that encloses all relevant regions of the scene, before a new virtual camera is created where all objects are located in front of its origin. After that, the view frustum is stretched to a unit cube with a perspective transformation of the scene and the shadow map is rendered. Similarly, light space perspective shadow maps [345] are calculated in post-perspective camera space for reducing aliasing artifacts by scaling up those regions, which are next to the near plane by transforming the camera frustum to a unit cube. But here the calculation of the virtual camera is done in light space, which allows treating all lights as directional lights, to alleviate the perspective distortion, which otherwise leads to a remarkable loss in quality for objects outside the close-up range as compared to perspective shadow maps. The virtual camera position thereby determines the strength of the perspective distortion.

In this regard, trapezoidal shadow maps [215] are well suited to easily parameterize the latter by adjusting the so-called focus region, which should cover about 80% of the shadow map resolution. Here, a trapezoid is calculated that encloses the view frustum from light

view and which later is transformed into a unit cube. Parallel-split shadow maps are another technique that especially focuses on big scenes [354]. The basic idea is the same as with trying to avoid z-buffer flickering in general, namely by splitting-up the view frustum in several bins by using planes parallel to the viewing plane and then processing these parts individually. An in-depth survey of real-time hard shadow mapping methods very recently was also presented in [275].

On the other hand we have filtering approaches like the so-called percentage closer filtering shadows (PCF) [257]. PCF simulates soft shadows – which in contrast to hard shadows are cast from area light sources (that do not exist in OpenGL) – by calculating the mean value of n shadow tests for every pixel. Because filtering the depth values in the same way as color textures would lead to nonsense distance values, this method averages the results of the depth comparisons instead, which yields a percentaged occlusion. For fixed kernel sizes PCF only requires Shader Model 2.0 functionality and is for small n already directly supported by modern GPUs.

Likewise, variance shadow maps [67] tackle the problem of filtering depth textures by storing the mean and mean squared of a distribution of depths instead of a single depth value per texel. This allows to efficiently compute the variance over the filter region, which is then used to approximate a fragment's occlusion. Perspective PCF soft shadows (PCSS) are a shadow mapping technique that renders perspectively correct soft shadows with a varying penumbra, which depends on the light source's size and its distance to an object [82]. PCSS thus also includes the distance between occluder and occludee by using a variable kernel size based on an estimate of the penumbra size given by the distances of blocker and receiver pixels to the light source (see Figure 3.7, right). Due to the higher complexity of this algorithm, PCSS shadows are slower and SM 3.0 is required.

The X3D lighting model [336] still does not include any form of real-time shadows since it follows more or less the old GPU fixed-function pipeline [174]. However, there have already been some proposals to include real-time shadows in X3D, since this feature is essential for future applications, also beyond the field of Mixed Reality (see next section). For instance Sudarsky [300] generates shadows for dynamic scenes, but only on user defined surfaces and for point lights that must be outside of the current scene. The *Shadow* node extension from Octaga [229] supports all X3D light types – *DirectionalLight*, *PointLight* and *SpotLight* – and provides a "detail" field parameter that is used to specify the resolution of the shadows. However, soft shadows are not supported, and the implementation as *Group* node, which links all occluders and lights, does not look very natural.

The BS Contact browser from Bitmanagement [26] provides vertex and fragment shaders, which can be used to generate dynamic shadows, but the application developer has to handle the shadow map generation and processing per object directly in the shader code. This may be a very flexible solution but not one which all scene authors can handle and would intuitively think of. Recently, Kamburelis presented another set of node extensions for integrating shadow maps and projective texture mapping into X3D [167]. However, the proposed extensions only deal with simple standard shadow mapping and – on a rather primitive level – PCF shadows. Furthermore, the proposed shader integration is very low-level and some of the proposed extensions even seem to be mutual contradictory. Nevertheless, despite their drawbacks some of the proposed extensions might ease

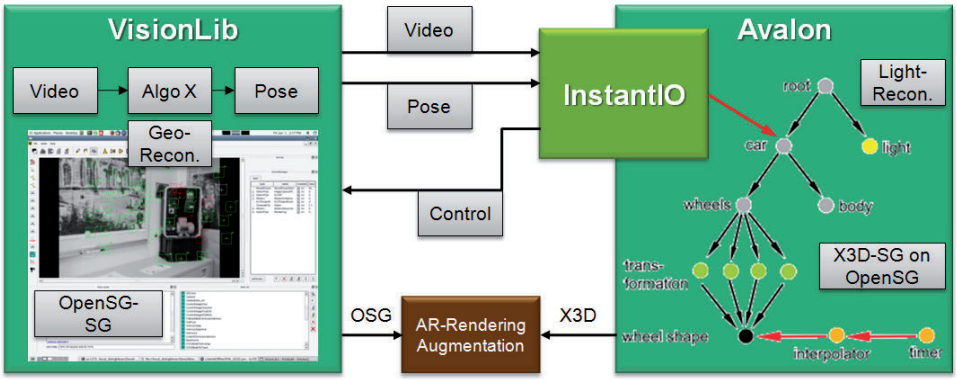

Figure 3.8: *Connection between tracking system (left) and rendering (right) in Instant Reality.*

improving rendering quality a lot and should be simple to use, since e.g. the new *GeneratedShadowMap* node in combination with the *TextureCoordinateGenerator* extension for calculating projective texture coordinates is rather close to the X3D design principles.

3.3 Lighting in Mixed Reality

In contrast to Virtual Reality (VR), in Mixed Reality (MR), which spans the whole continuum between Virtual and Augmented Reality (AR) [137], compare Figure 1.2, we do not have completely synthetic scenes. Instead, we try to integrate virtual objects into existing real scenes. To do this, we have to determine the exact pose of the user, which is usually done by using video tracking systems. Furthermore, we have to put a video image of the real scene behind the virtual objects. Image analysis in general and tracking in particular are out of scope research topics not covered by this thesis, but nevertheless we first have to consider interfaces to integrate various sensor data streams from external devices like tracking systems and video cameras into the scene graph [52]. As shown in Figure 3.8, in the Instant Reality framework [135] this connection is handled by the "InstantIO" subsystem, which is described in more detail in [51].

To seamlessly integrate virtual objects into the real scene at interactive frame rates, it is also important to employ advanced rendering techniques. One problem is how to reconstruct and render the lighting fast enough for being real-time capable. Therefore, a classification of illumination methods based on their input requirements, i.e. the amount of radiance and geometry known from the real environment, was presented in [137]. Figure 3.9 (left) shows a typical AR application, and in Figure 3.9 (right) a screenshot of a pre-rendered AR application is shown (compare [298]), where shadows are used to provide depth-cues but which do not correspond to the real position of the sun. As can be seen, for a better integration into the real scene, the virtual objects must be lit in a realistic way, and they have to cast shadows onto real objects etc. [95].

To achieve this, the synthetic objects not only need to be registered geometrically but also photometrically for consistent lighting. This is usually done by means of image-based lighting (IBL) techniques for computing the lighting by previously acquiring a high dynamic range (HDR) light probe image or by using a 180 degree fish-eye lens for

Figure 3.9: *Left: typical Augmented Reality application showing some simple Gouraud shaded augmentations and text. Right: screenshot of an interactive pre-rendered AR infotainment application for explaining how "The Messel Pit" originated.*

capturing real time environment maps. Lastly, the effects of changing the light transport paths of the real scene by inserting synthetic objects (which, for example, can be occluded by real objects or throw shadows onto them) also have to be taken into account. This can be solved with differential rendering [58], a rendering technique that adds the difference of two synthetic images to the real input image (cf. Figure 6.17, p. 195).

Consistent illumination in AR applications is still an open field for research because besides the lighting simulation, i.e. photo-realistic rendering, three other problems have to be solved in advance: geometry reconstruction for handling e.g. occlusions; second, lighting reconstruction for recovering number and type of primary light sources (which eventually are the cause for shadows as well). The third problem is the material reconstruction for determining the reflectance properties, like the BRDF and the BTF, of real materials, which is quite essential for computing correct inter-reflections and shadow color, but in the general case beyond the scope of this thesis. Likewise, geometry reconstruction is out-of-scope here. Though occlusions can be directly handled by integrating time-of-flight cameras [163], reflection properties cannot be obtained this way.

In [58], differential rendering was presented in order to merge virtual objects with real scenes. It is a two pass composition technique that is feasible for augmenting images or video streams with consistent illumination. It requires two lighting simulations, one with the real scene only and a second one with additional virtual objects taken into account. The term lighting simulation usually implies global illumination techniques, therefore the first implementations were radiosity based (i.e. hierarchical radiosity, with a line-space hierarchy of links and light shafts for rapidly identifying modified links when interacting with the virtual object [69]), because of its view independence and good quality.

This approach of Drettakis et al. was improved by Loscos et al. [206] by introducing a special pixel data structure with a final gathering pass for handling the direct lighting component. Although they did not deal with moving viewpoints and light sources, animations or even complex BRDFs, the use of global illumination methods has the advantage of automatically computing shadows and indirect illumination, which for example is observable in reflections and color bleeding effects. Thus, in [107] this method was extended to also handle reflective and refractive objects correctly.

Although global illumination is much too slow for realtime applications, it is still prevailing in the field of consistent illumination. An interactive ray tracing approach was proposed by Pomi and Slusallek in [250], but it needs special ray tracing hardware. Real-world lighting information for shading virtual objects was first introduced by Debevec [58]. He distinguishes between the local scene, for which a reflectance model is needed, and a distant, light-based scene, which only serves as the source for natural illumination. This incident lighting information, the real scene radiance, is captured by using an omni directional HDR image [61] of a mirrored ball, the so called light probe. He also states, that the realism of the final composited image depends on the error in the rendered local scene compared to the real image and hence on having a good model of the reflectance properties of the real materials.

Image based lighting and irradiance maps are used in mixed reality simulations as a means to transfer real world lighting onto surfaces of virtual objects. Illumination of the real environment is passed to the virtual object via irradiance maps from a light probe, captured as HDR images. In [173], the authors discuss ways of filtering environment maps to create different types of irradiance maps. Spherical harmonics (SH) are used to extract the diffuse frequencies of the environment map, while hardware generated mipmaps are used to create glossy maps. A combination of the original and both filtered images yields the final reflection that is mapped onto the virtual object. In [179], the author proposes to use irradiance maps in conjunction with ambient occlusion (AO), a statistical method that determines shadows under complete diffuse lighting conditions, without considering any light sources [245]. These values can be used to attenuate colors from the irradiance maps to simulate self-shadowing under the assumption of distant global illumination.

A GPU based real time method was proposed by Gibson et al. [94, 96], in which the virtual object is illuminated by using an irradiance volume in combination with cubic reflection maps. Another important aspect in this context is the final tone mapping, in which the radiance is transformed into displayable pixel colors in the range $[0, 1]$ by means of the camera response curve [61]. Shadows are computed by utilizing a shaft-based data structure between source and receiver patches in the sense of a radiosity system with pre-computed transfer of the radiance along these light-, and therefore potentially shadow-, paths. An interesting approach combining geometric and photometric calibration was recently outlined by Pilet et al. [246]. It uses a calibration object with known shape and normals and view independent albedo from which a light map is calculated based on Lambert's law and the observed mean intensities.

Another framework using IBL and soft shadows (but not HDR), can be found in [303]. Image data is captured via a 180 degree fish eye lens or photos of light probes. Because these images contain direct and indirect lighting, and the positions of the light sources cannot assumed to be readily available from the footage, they must be filtered to extract different frequencies of lighting for different types of material, for instance low frequency lighting for diffuse surfaces. The authors use an image space blur-filter to generate the different irradiance maps. A downscaled mipmap is used for very low frequencies. While this approach is fast and stable, the deviation from a correctly filtered image becomes greater with decreasing frequencies. Another important aspect is, that for specular irradiance maps, there is no clear connection between the radial blur and lighting parameters like the shininess as given in the here used phong lighting model.

In [256], an image-based lighting technique with irradiance environment maps is presented. The authors use the spherical harmonic basis to filter spherical images and reuse the low-frequency spherical harmonic representation to simulate diffuse reflection via environment mapping. In [288], Sloan et al. introduce precomputed radiance transfer with the spherical harmonics basis and describe the mathematical tools to precompute light transfer functions, simulating diffuse unshadowed, shadowed and self-reflected surfaces. The major insight here is that a shortened version of the rendering equation (see equation 3.13) – similar to the well-known Fourier transform – can be evaluated in real-time by dotting the transfer and light coefficient vectors.

In order to cast realistic, dynamic shadows onto real as well as virtual objects from reconstructed or even virtual light sources (e.g. additional lamps), appropriate shadowing algorithms must be integrated, too. Suitable real-time shadow techniques are discussed in more detail in section 3.2.4. Recently Egges et al. [74, 240] presented a mixed reality framework for virtual characters that builds upon VHD++ [251]. For rendering, a dynamic PRT approach (dPRT) also suitable for virtual humans is proposed, but it heavily relies on precomputation and manual work for defining special receiver and occluder objects, requires static lighting conditions given as light probe image, and also leaves subsurface scattering effects aside. In addition, [240] provides a comparison of MR illumination models concerning their usability for deformable virtual humans.

Because correct shadows are essential for a proper perception, more algorithms are needed to identify direct lights inside the irradiance maps. Therefore, in [303] and [190] some methods are shown for retrieving the number and 3D positions of the light sources, but they still all have to deal with problems like processing time and jitter concerning the derived light direction and area. In [60], a median-cut algorithm for extracting light sources from a given environment map was proposed, including their color and intensity (compare Figure 6.8, page 175). This method later was adopted for real-time augmented reality applications [209], in order to approximate the lighting situation by means of a finite set of geometric light sources.

3.4 Human Hair

Hairs belong to the group of keratin fibers and consist of three layers [292]. The hair cuticle resp. outermost hull consists of overlapping squamous cells, which is shown in Figure 3.10 (right). The middle layer is the cortex, which surrounds the porous, inner most layer of the hair shaft, the medulla. Like skin, the color of hair depends on the pigment melanin that is stored in the cortex cells. Common forms are pheomelanin, which causes red hair, and eumelanin, which is responsible for the other colors. Hairs can be further divided up into the hair follicle, which is the part located beneath the skin (see Figure 3.12, left), and into the aforementioned hair shaft above the skin surface.

A human being usually has around 100,000 hairs, though grown-up males often have a reduced hair density due to hormonal influences. The hair length depends on the type of hair. In contrast to terminal hair that develops during puberty on certain body parts depending on the gender (e.g. a beard), the fine vellus hairs are barely longer than 1 mm. They cover almost the entire area of a child's body as well as most parts not covered by

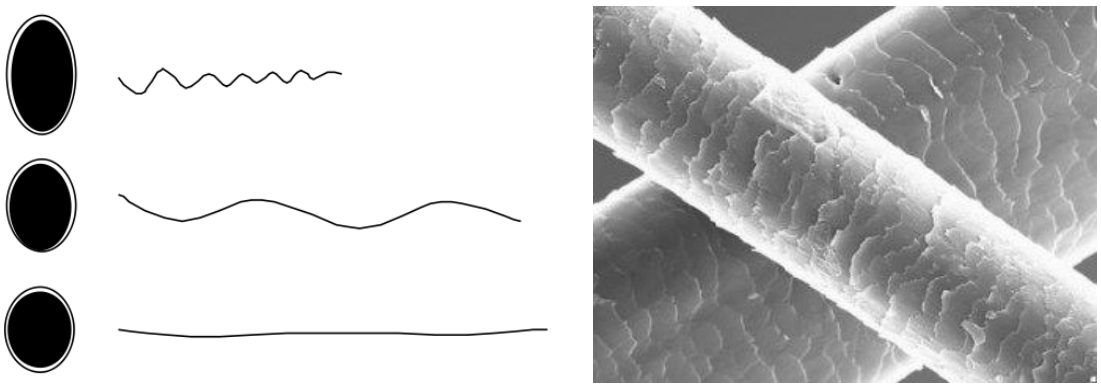

Figure 3.10: *Left: visualization of the relationship between waviness and ellipticity of hair (cf. [293, p. 10]). Right: hair under the electron microscope.*[4]

terminal hair of grown-ups and help regulating the body temperature in that sweat can more easily leave the skin pores. As can be seen in Figure 5.3 (left) on page 140, this type of hair is especially of interest for skin rendering [184].

Due to differing stages of hair growths, the scalp hair can reach a size of up to 60 to 100 cm, whereas the diameter of a hair strand can vary from 0.017 to 0.18 mm. As visualized in Figure 3.10 (left), the waviness depends on the shape of the hairs: the higher the degree of ellipticity, the more curly the hairs are [293]. Concerning mechanical properties, strain and shear strain and, even more important, bending and torsion are influencing factors for the deformation behavior of hairs. An important parameter here is Young's modulus of elasticity E, which differs for both principal axes because of the elliptic form of hairs (with $E \approx 3.79 \cdot 10^5 N/cm^2$ [293]). For small deflections bending can be calculated analogously to the deflection of a beam with a fourth-order differential equation. Because of twist, curliness can be described with the torsion modulus G. Another factor is friction, which is anisotropic due to the squamous structure (Figure 3.10, right).

3.4.1 Modeling and Simulation

In order to create convincing human hair there are basically four problems to solve: modeling and styling, hair dynamics, collision detection and response, and finally hair rendering [53, 226, 330]. Though many DCC tools, like Maya [14] or Cinema4D [216], provide plugins for hair modeling, the resulting model requires special algorithms and is only suited for offline rendering. Another approach allows interactive hair modeling based on color-coded textures [126]. By painting into a scalp image that is spanned along its uv-coordinates, parameters like position, density, length, and color can be specified. Although the focus does not lie on modeling, a short review of common hair models is necessary beforehand, because not every hair model is suitable for every animation and rendering method. After that we analyze existing approaches for hair simulation, whereas many of them are also only suitable for offline rendering.

Presently a seamless transition between those four categories is still problematic because

[4] Taken from http://graphics.stanford.edu/~mcammara/hair/

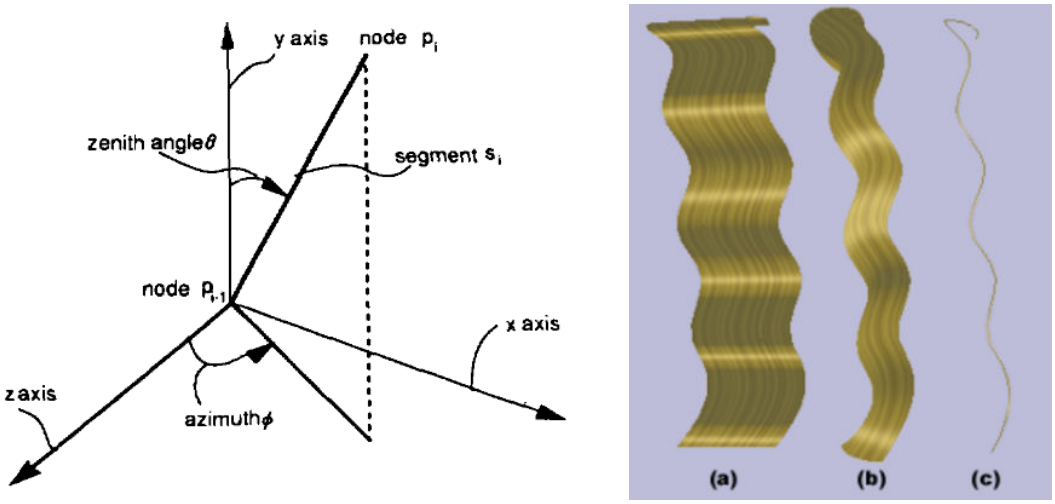

Figure 3.11: *Left: polar coordinate system for animating a hair segment (taken from [131]). Right: LOD representation of hair [333]: a) Strip, b) Cluster, c) Strand.*

the fewest hair simulation systems are self contained and they all differ in their geometrical representations, animation methods, and lighting models. Furthermore many approaches focus on special hair styles like fur or short hair [101, 108]. Thalmann et al. [226] classify hair models into several categories. The first one contains explicit models, which are relatively intuitive but computationally expensive, because every single hair strand is considered individually [131, 53]. The next category are the cluster models, which utilize the fact that neighboring hairs have similar properties and tend to group together. Cluster models can be further divided up into hierarchical and flat models.

Ward et al. [331, 332, 333] propose a hierarchical level of detail representation, in which the hairs are either represented as precomputed strips, clusters or strands respectively (see Figure 3.11, right), depending on their visibility, distance to the viewer and velocity. With the help of a quadtree the base hairs, i.e. the strands that resemble the coarsest resolution level, are then subdivided until their width is below a given threshold. In this case the next finer resolution (the cluster or cylinder) is chosen, and the recursive subdivision is continued again, until the finest level made up of single strands is reached. Simulation is done via a set of base skeletons following [131], whereas during runtime the quadtree is traversed according to the aforementioned criteria. Collision are handled with a bounding volume hierarchy of swept sphere volumes.

More common are non-hierarchical schemes in which hair clusters are represented by generalized cylinders [178], trigonal prisms and polygon strips. The latter are rendered by using parametric surfaces in the method suggested by Koh and Huang [185]. Particle systems, especially the loosely connected particles method proposed in [17], can be seen as an extension to clusters. Here, the particles serve as sampling points for tracking hair dynamics in order to overcome the tight cluster structures during lateral motion. Volumetric textures are primarily used for very short hair [165]. The last category regards hair as a continuum that behaves either like a fluid (for lateral motions) or like a solid body [111]. Hair shape modeling is done by placing some ideal flow elements, whereby collisions are implicitly avoided by the use of streamlines.

In [350] a method is suggested for adding curliness by modulating the hair strands with offset functions. In the model of [191], a hair patch is composed of quadrilateral segments, and curly effects are achieved by projecting the vertices onto an imaginary cylindrical surface. To simulate complex hair styles in real-time, in [322] a method is proposed in which an FFD grid that contains the hairs is being deformed by treating the grid points as mass points of a mass-spring-system. Bertails et al. [25] presented a physically-based model, where each hair is represented by a so-called super-helix. This also allows simulating curly hair but is not suited for interactive applications. An impressive real-time approach for simulating smooth hair styles, which runs completely on the GPU and also considers hair-hair collisions recently was presented in [306], though this method requires very modern high-end graphics hardware.

Animation can be done via key-framing or vertex perturbations on the one hand and numerical simulations on the other hand, reaching from mass spring systems over free form deformation and rigid multi-body chains up to vector fields [111], based on the underlying hair model. A computationally rather cheap animation method also based on differential equations is the modified cantilever beam simulation originally proposed by Anjyo et al. [131] for computing the hair bending of smooth hairstyles during the hair shape modeling process. In their model, a hair strand is modeled as an open, serial kinematic multi-body chain with anchored root and segments that are connected though joints. As visualized in Figure 3.11 (left) the strand then is deformed by obtaining the new angular positions (Θ, Φ) of each control point.

The main drawback of most simulation methods is the fact, that depending on a hair style's complexity the simulators often cannot guarantee for the hair style's preservation, and for simplicity they often neglect the existence of head and body movement in collision detection. But collision detection and response are a fundamental part of hair simulation. Absolutely inevitable is the treatment of hair-head collision. Whilst geometry traversal, hierarchical or grid based schemes, and vector fields offer more precision, for real time applications a head can be approximated sufficiently with the help of parametric bodies like spheres or ellipsoids [131, 185, 224].

Hair-hair collision for interactive applications is still mostly ignored or quite coarsely approximated by bounding volume hierarchies, by density distributions [188], by inserting some extra spring forces between neighboring strips [185], or by generating auxiliary triangle strips between horizontally neighbored guide hairs against which collision is tested, and whose links can be broken or recovered dynamically based on their distance [39]. Lately, in [352], inter-hair collisions are handled by first voxelizing the hair volume as well as the character's mesh, and by then pushing vertices falling in high density areas into the direction of the negative gradient of the voxelized density field.

3.4.2 Hair Rendering

Last but not least rendering itself covers the full range from drawing single-colored polylines [126] and alpha-textured polygonal patches [185] over heuristic local lighting models for anisotropic materials [276] up to physically and physiologically correct illumination solutions [213]. Based on measurements of hair fibers the latter explains the existence of

a second specular highlight visible by light hair by means of light paths described by a transmission-reflection-transmission-term *TRT* and simulates lighting with expensive ray tracing. In contrast the first two rendering methods are quite fast but not very convincing, because they don't take anisotropic reflection and self-shadowing into account, which are regarded as the most important lighting effects of hair by Koster et al. [191].

The latter can be achieved with the help of an opacity map [177], which discretely approximates the light intensity attenuation function for encoding a fragment's opacity value, and which are created from a light source's point of view (mostly in a preprocessing step [191]). An improvement where the opacity layers were adapted to the form of the hair volume recently was proposed by Yuksel and Keyser [351].

An often referred reflectance model for dark hair, which exhibits higher reflectance than transmission and almost no self-shadowing, can be found in [165], and was extended for backlighting effects in [101]. Based on Marschner et al.'s [213] results, Scheuermann [276] extended this lighting model for the use in hardware shaders by perturbing the hair tangent for shifting both highlights (see Figure 4.21, page 126). To overcome this more phenomenological approach, where multiple scattering is faked with an ad-hoc diffuse component coupled with transparent shadows, Zinke et al. [355] recently presented the concept of dual scattering, which splits the multiple scattering computation into a global and a local multiple scattering component.

Problems when rendering hair as polylines or curves as e.g. in [224] are aliasing and the fact that the hairs have the same width, irrespective of the distance to the viewer, which requires additional filtering. In addition, antialiasing of line drawing as a special case and alpha blending in general needs correct visibility ordering. In [276] the blending problem is solved by drawing the appropriately pre-sorted hair patches in several passes. Kim and Neumann [178] suggest an ordering algorithm in which each line segment belongs to a slab perpendicular to the viewing vector subdividing the hair volume's bounding box in order to simplify the process of back to front rendering.

A very comprehensive overview on recent techniques is also given in [330]. But in summary, all these models usually only deal with certain aspects of hair modeling, simulation and rendering, yet almost none of them is simply adoptable to our needs in that an easily parameterizable, robust, stable, fast and convincing simulation method is taken into account as well as realistic real-time shading for the use in complex virtual environments.

3.5 Skin and Emotion Visualization

Modeling and rendering human skin is an important and still open topic in computer graphics, and to visualize it, several phenomena have to be considered. In addition, attitude and emotional state as well as pragmatic and discursive elements are communicated through head movements, gaze behaviors, and facial expressions. The latter results from subtle muscular contractions and the human eye is extremely attuned to and familiar with them. Hence, creating and animating 3D faces is an important research topic especially for the film and game industry. Besides the advances in motion capturing techniques, which can be accurate enough to capture even slight movements in emotional expressions,

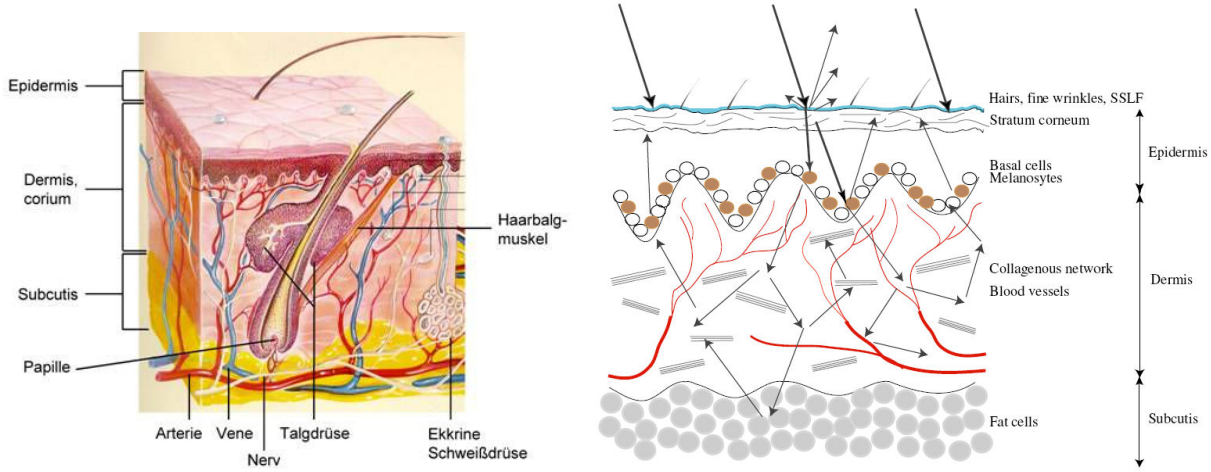

Figure 3.12: *Schematic view of the multi-layered structure of human skin (left),*[5] *and optical path of light within the skin (right, taken from [133, p. 31])*

new rendering techniques ensure realistic skin models, too.

The human skin is the largest organ and an extremely complex system that is deeply connected to the entire body. It is responsible for regulating the temperature, controlling water loss, and also protects the body (muscles, internal organs etc.) from external microorganisms and the like. Skin appearance can be classified according to spatial scale. The micro scale is defined by cellular level elements like collagen fibers and organelles whose dominant effects are scattering and absorption. At this level, skin exhibits a very complex structure consisting of three major layers (see Figure 3.12, left).

The top-most skin layer is the stratum corneum, which belongs to the epidermis and which is about 0.01 to 0.04 mm thick. Because of its scaly, wrinkled structure with dead cells, pores, and spots, light is reflected diffusely. Besides greasy secretion and moisture the optical properties are also characterized by the fine vellus hairs that can be seen as sparse point scatterers, and which are responsible for the so-called "asperity scattering" [184]. This is visible as light sheen especially at contours (cp. Figure 5.3, page 140), though most light incidents into deeper layers.

The epidermis is the outermost layer with a thickness of about 0.1 to 1.5 mm and contains no blood vessels. It is a semi-transparent, non-homogeneous optical medium that is mainly colored through the pigment melanin (cf. section 3.4). The skin tone is further influenced by the slightly olive bilirubin and the yellowish beta-carotene [299].

The dermis is about 0.6 to 4.0 mm thick and houses the hair follicles, sebaceous and sweat glands, nerves, and the blood vessels that causes its red appearance. This layer can be further divided up into papillary and reticular skin and is riddled with thinner, papillary and thicker blood vessels. Whereas light in the epidermis particularly is absorbed by melanin, in the dermis it gets scattered by collagen fibers until it is finally absorbed by the hemoglobin, which is responsible for the reddish coloring.

According to [299], collagen fibers with a diameter of around 0.003 mm are responsible for Mie scattering, where the size of a cellular element is close to the wavelength of light, and

[5] See http://edoc.hu-berlin.de/dissertationen/auwaerter-volker-2006-01-20/HTML/chapter2.html

fibers and micro structures that are much smaller than the wavelength of incident light cause Rayleigh scattering. The latter causes shorter wavelengths to be scattered more than longer ones [138]. The last layer is the subcutis, whose thickness largely depends on the body region, and which includes blood vessels, nerves, and lots of white fat cells that reflect incident light back into the upper skin layers [133].

3.5.1 Skin Rendering

Visualizing virtual characters is an important task to create immersive virtual environments, convincing digital story settings, and games. When rendering human skin, several phenomena have to be considered, e.g. the fact that human skin consists of several layers that are slightly translucent, because cells contain large quantities of water [115]. Therefore, skin does not simply reflect light like plastics or metal. A thin layer of hair and moisture on the rough and oily skin reflects only small amounts of light, while most of it travels into deeper layers of skin, which e.g. causes bright light shining through the ears or fingers and gives skin its soft appearance.

While the 3D meshes for the virtual characters have to be designed and crafted in a pre-production phase, many of the accompanying (often dynamic) phenomena, including their timing, order, and intensity, are computed during runtime. Another issue only barely considered in skin rendering is affective aspects. Temporal variations of facial coloration because of blushing and paling, along with sweat and tears are important in order to simulate consistent, believable expressions for a virtual character [166].

Skin is a multi-layered non-homogeneous medium with translucent layers that have subsurface scattering properties [133]. The colors of facial skin are mainly due to the underlying anatomic structures: muscles, veins, and fat all are visible through the skin's translucent layers. In this regard, subsurface scattering is one of the most important phenomena of human skin. It describes how light is scattered when it enters a partly transparent medium. Here, light is not simply reflected but incidents more or less deeply into the material, scatters inside, and then leaves at different positions including those possibly not lit by a light source at all (see Figure 3.12, right). This can simply be observed by holding a bright flashlight behind the fingers.

There are several ways to simulate subsurface scattering effects, but most of them are offline methods. E.g. Hanrahan and Krueger [115] presented a computational model for subsurface scattering in layered surfaces such as biological tissues, as a simplification of the general volume rendering integral (cf. [80, page 8 ff.]). To account for both, Mie and Rayleigh scattering, in the model of Hanrahan and Krueger the material phase function for representing the directional scattering of incident light is described with the empirical Henyey-Greenstein function [125]. Depending on a parameter g (the mean cosine of the scattering angle) the incident light is scattered isotropically (for $g = 0$), with predominant forward scattering ($g > 0$), or backwards ($g < 0$), whereas skin is anisotropic with significant forward scattering ($g \approx 0.85$ [138]).

With PRT, Sloan [288] has presented a technique that is real time optimized, but needs intensive pre-computation and is furthermore limited to fixed geometry. Several extensions [289] have been made to the original algorithm. However, this approach is not suitable for

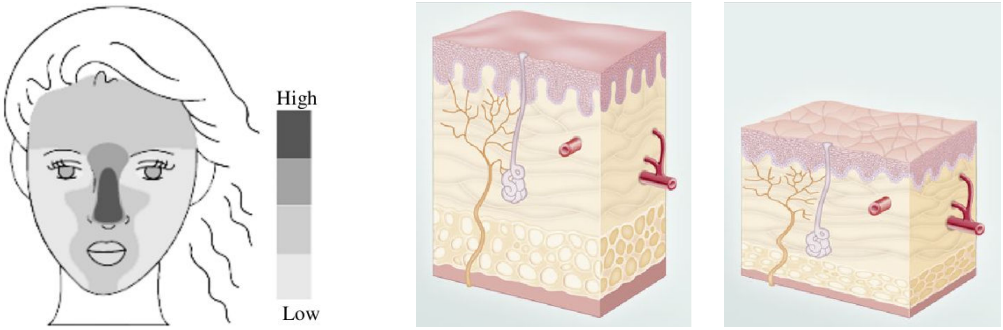

Figure 3.13: *Left: the appearance of skin differs at the meso scale, e.g. the so-called T-zone has lots of sebaceous glands [133, p. 26]. Right: comparison of young and old skin.*[6]

simulating subsurface scattering for varying lighting situations and interactive characters with highly dynamic geometry that is manipulated, possibly in vertex shaders, through morph targets etc. Radiance transfer thus cannot be precomputed and the memory overhead is very large. Although with dPRT in [240] an approach for deformable objects was presented (see section 3.3), the problem of subsurface scattering was not tackled at all.

Other rendering methods consider the aforementioned layers as layers of color – for instance a muscle map can be used to give skin its pinkish complexion within a layered texture shader. Based on an offline skin rendering approach [31], which utilizes the fact that for diffuse reflectance the outgoing radiance is spatially blurred, in [269] a technique is proposed that approximates subsurface scattering on a still more phenomenological level by blurring the diffuse illumination in texture space using graphics hardware to achieve real-time frame rates. To further emphasize light bleeding in case of backlighting, so-called rim lighting is applied by additionally adding the dot product between light and view vector that is scaled by a Fresnel term.

Yet another approach is the extension of the commonly used BRDF by measured skin data (e.g. [195]) and a real subsurface scattering part. The resulting BSSRDF is presented by Jensen et al. in [138]. He proposed a method to split subsurface scattering into a single and multiple scattering part, whereas the latter is mostly shown by materials like skin or milk. Following [115], the presented approach is derived from the volumetric behavior of light propagation in a participating medium, which can be described by the volume rendering equation, and the observation that the light distribution in highly scattering media becomes isotropic. The complete BSSRDF then is a sum of both scattering terms. With limitations, mainly by only taking local scattering into account and evaluating the equation in image space, the model can be used for real-time environments [219].

Another method, more practical for real-time application, is based on maps recording the distance of a point seen from the light source. This depth map is used to measure the covered distance of the light ray inside a given geometry, and can be regarded as an approximation of path tracing. Such a technique of approximating extinction and in-scattering effects is presented by Green [106] and is based on an offline approach for Pixar's RenderMan system described in [127]. Additionally an attenuation look-up texture can be used that maps the normalized light distance to a color ramp – e.g. from white

[6] See http://www.focus.de/gesundheit/ratgeber/haut/interaktive-infografik_aid_137702.html

(direct reflection) over amber and red (to account for epidermis and dermis respectively) up to black (i.e. full attenuation). Besides the limitation that only homogeneous materials without media transitions are considered, only the first object in sight is recorded to the depth map, which can cause artifacts.

A good overview on real-time, GPU-based skin rendering techniques can also be found in d'Eon and Luebke [64]. For simulating scattering, the authors further present a method that combines the aforementioned texture space diffusion approach [31, 269] with modified translucent shadow maps [49] (similar to Green's method [106]). Here, texture space diffusion is done with the help of sum-of-Gaussians diffusion profiles for a three-layer skin model. In this regard, a diffusion profile $R(r)$ describes for all color channels, how the incident light attenuates for each distance r around a surface point.

For rendering, each diffusion profile is first approximated by a sum of six Gaussian functions as pre-process. Then, the diffuse illumination is convolved (or blurred) in texture space for each Gaussian kernel. After that, the blurred textures are combined according to $R(r)$. Likewise, Jimenez et al. [139] recently proposed a scalable approximation method for subsurface scattering, which is intended for game contexts and produces translucency effects at high frame rates. The main difference to the previous approach is the idea to simulate scattering in screen space as a post-process instead in texture space.

Exploiting the observation that the effects of subsurface scattering are more noticeable on small curved objects than on flat ones, in [198] subsurface scattering is simulated using a curvature-dependent reflectance function and a local illumination model. Though lighting is evaluated on a phenomenological level, less pre-processing than in [64] is required and rendering is only around 5% slower than Blinn-Phong as no multipass is needed.

On the macro scale skin is divided up into body regions like nose, cheeks or torso and can easily be represented by polygonal geometry. Depending on these regions the appearance of human skin at the meso scale differs; e.g. the so-called T-zone (as shown in Figure 3.13, left) appears glossier because more lipids are secreted. But this depends on age, gender, or race, and varies from one person to the other, and hence can't be generalized. For example the skin of an elderly person appears less translucent than a young one since very little lipid is stored on the skin surface and therefore less backscattering occurs.

Skin features like tissue cells, surface lipids, hairs, freckles, pores and wrinkles are often omitted for rendering or handled by texture and bump maps. Although direct surface reflection only happens at the stratum corneum, and even there only for approx. 5% of the incident light, the resulting reflectance is significant for a realistic appearance. Here, the reflected light does not get colored by specular reflection, since skin is a dielectric material [115, 64], and therefore this can be handled by the Fresnel equations (with the indices of refraction $n_{air} \approx 1.0$ and $n_{skin} \approx 1.4$).

Because skin is a heterogeneous material and its appearance also varies with viewing and illumination direction, the usage of bidirectional texture functions seems appropriate, but current data acquisition methods are not only expensive and very time consuming, but at the moment they do not work correctly for in vivo taken images [133].

As skin ages, it becomes thinner, and the amount of melanin, collagen and elastic fibers, as well as the number of perspiratory glands decreases (compare Figure 3.13, right). Because perfusion also decreases, the skin appears grayish, though more noticeable are

skin relaxation and the appearance of wrinkles and age spots. The latter is caused by an irregular production of melanin as well as an exceeding production of the yellow-brown pigment lipofuscin [133]. Methods for simulating aging based on a database of 3D scanned faces on which learning is applied were proposed by [234, 274], though the costs in terms of example face data and computing time are rather high.

3.5.2 Psycho-physiological Factors

The human face can be seen as a system that communicates with various types of signals like muscular activity [166]. Faces do not only have static features such as skin color and feature size, or slow signals like the development of wrinkles, but they also exhibit rapid dynamically varying signals, which are often caused by emotions like joy or sadness. This also includes the simulation of tears, rendering of wrinkles, and skin tone changes.

3.5.2.1 Psychological and Medical Foundations

Whereas emotional variations of gestures and mimics have been subject of extensive research, a more unattended field are psycho-physiological processes like the change in skin color, which can occur when an emotion is very strong. In this context basically two types can be distinguished: Blushing is an uncontrollable reflex with a duration of about half a minute which usually occurs in a social context, when a person feels ashamed or embarrassed. Here regions with many blood vessels like cheeks, ears and forehead, get more reddish. Still no physiological or computational model exists for this human reaction. The same is true for pallor which occurs when feeling fear or pain. The face gets pale because the blood is redirected to more important regions.

Not only speech and gestures but also turning pale or blushing (as one of the "most human of all expressions" according to Ch. Darwin, 1872 [238]), which like gestures and mimics is part of the non-verbal communication, can be important within interpersonal communication. In contrast to the central nervous system, which for real humans is responsible for the conscious control of motor functions, the autonomic nervous system (ANS) controls unconscious inner functions that can result in physical reactions like blushing, pallor and similar phenomena, which sometimes are described by the term vascular expressions.

But except for standard idle behavior like slight movements as well as blinking and sometimes breathing, other unconsciously happening behavior, specifically physiological symptoms like crying or sweating that are controlled by the autonomic nervous system (ANS) until recently were mostly left unconsidered. Although these symptoms of very strong emotions can be a crucial component of a realistic look and feel (Figure 3.14), for emotionally caused skin color changes and other physiological phenomena there still neither exist approaches similar to FACS nor any other parameterization model. Thus, for being able to dynamically parameterize skin changes caused by emotions based on physiological and psychological knowledge we first need to survey the corresponding literature, too. A short overview on current emotion models can be found in section 2.1.3.3.

The facial skin is a very strongly perfused region of the human body. This is controlled by the autonomic nervous system [291], which is independent of the central nervous sys-

Figure 3.14: *Complexions (from left to right: red cheeks, red head, neutral color, pale face)*[7].

tem (CNS). The autonomic nervous system controls the unconscious inner functions in the human body (cardioplegia, respiration, blood pressure, etc.). In contrast to the peripheral nervous system, which in combination with the CNS processes all sensations and deliberately controls the motions of the limbs, people can not directly control or influence body functions that are controlled by the autonomic nervous system. There exists a relationship between the autonomic nervous system and the emotional feeling. This results in physical reactions like red spots, blushing, paleness, sweating or weeping.

Blushing [166, 211] in the face is a striking expression of emotions. It can occur when a sensation is very intense. But blushing is not just an expression of emotions. It also occurs during physical effort. Thus, this is further distinguished between blushing (due to psychological reasons) and flushing (due to physiological reasons). Blushing is a vascular effect and belongs to the vegetative physiological changes of the body. Blushing creates a feeling of warmness. It mostly arises for light-skinned people, because their facial skin has only a slight pigmentation. Therefore, fine vasculature are visible. Blushing is an involuntary reflex, but it requires a certain social context where one draws attention from others. It most likely occurs during embarrassment and shame.

Regions for emotional reactions like blushing vary from person to person. However, usually the cheeks are affected by blushing for almost all persons: In a study of Shearn et al. [284] it was found that there is a correlation between blushing and increase of temperature in the cheek region. This was also found in [211] and further that it is the forward facing upper cheek region. The lateral cheek region is only half as strongly affected. An average blushing takes $\Delta t = 35$ seconds. After 15 seconds, it has the strongest intensity and then it decreases again. The face and the ears are also able to blush.

Though facial expressions override other forms of expression on the face, it can be ambiguous if consistent coloration does not coincide with the shown emotional expression. Following Ekman [76], his basic emotions can be discriminated according to a decision tree. Anger, fear and sadness are accompanied with a high heart rate. But whereas for anger the skin temperature is high, for fear and sadness it gets low. Happiness, surprise and disgust come along with a low heart beat rate. Here it was shown that in the case of disgust the skin temperature also decreases.

In the case of fear people often get pale. Pallor is caused by a reduction of blood flow in the face and circulation to the brain [166]. In the literature, this phenomenon of the

[7] Images courtesy of Patrick Gebhard, DFKI.

skin is mentioned infrequently. Pallor, however, has similar properties such as blushing. Regions, which tend to blush, can also get pale, but here the skin temperature seems to decrease. Beads of perspiration are likewise an expression of emotion in the face (e.g. due to fear). They also occur with physical strain. The perspiratory glands are controlled by the autonomic nervous system, too.

Other vegetative functions that are controlled by the ANS, or more precisely the parasympathetic nervous system, are weeping (sometimes accompanied by blushing) and sweat. Tears consist of proteins, enzymes, lipids etc. and help to protect the eye from infections. A dependent phenomenon of crying is an increased flow of blood to the head. Whereas infants, which are not relevant in the context of dialog systems, mostly cry to attract attention, adults usually cry due to certain events or moods such as grief, anger, and joy. But this further depends on culture, gender (young women cry more than men), personality, physical state, hormones, social factors, and the whole situation (e.g. the likelihood of crying is greater when being alone, though it can be used to manipulate others), and probably serves a communicative function [320].

3.5.2.2 Emotions in Computer Science

Human emotions are an important element in a communicative situation and thus should also be modeled to achieve plausible virtual characters with consistent behavior (see also section 2.1.3.3). Here, the more manlike a virtual character gets, the more people expect emotional behavior. Therefore, models that can be used for automatically synthesizing plausible communicative behaviors need to be considered, which include deliberative and reactive behaviors as well as physiology simulation and emotions. A comprehensive overview on computational models of emotion is given in [214].

In this regard, it can be distinguished between encoding models that fall back on insights from cognitive science, such as models of how humans process emotions, and decoding models that specify an intended effect. The latter may also use cognitive models, but they are mainly based on perception experiments etc. Whereas AI mostly deals with the first type, for graphics usually the decoding models are important. For instance, in the study of Buisine et al. [32], it was shown that concerning decoding performance the most effective combination of speech and emotional expression is to temporally position the facial expression at the end of an utterance, while visual realism is perceived higher when the expression is shown during speech instead.

Gestures and mimics reflect emotional behavior. But whereas posture and mimics both have been subject of extensive research, a more unattended field however beyond standard mesh-based animations is the change in color etc., which can occur when an emotion is very strong. But usually only body functions controlled by the central nervous system (CNS) like voice and motor response are considered in computer graphics by generating or playing-back different body animations and facial expressions.

The latter are well understood and categorized by psychological models like the well-known FACS (cp. section 2.1.3.3), though communication effectiveness is still an issue in virtual agents research [32]. Albeit with the help of modern graphics hardware the more subtle changes concerning face coloring can be covered, too, in computer graphics

currently not much attention is paid concerning this topic, although it was shown that correct coloring and texturing can enhance the perception of certain emotions [79].

Blushing and pallor can be achieved by e.g. blending color values with every vertex along with its position and normal. Therefore, in [243] a system is proposed, where the facial coloration is realized by changing the vertex color according to its position in the local coordinate system of the head. The amount of colorization is controlled by the emotional state of the virtual character. Alternatively, similar to a bump map for simulating wrinkles a blush texture map can be used as originally proposed by Kalra and Magnenat-Thalmann [166], where another facial animation system based on predefined image masks defined by Bezier patches for creating texture maps is introduced.

Although being quite outdated concerning their rendering methods the prime contribution of these works was to point out that realistic skin rendering also requires considering changes of skin color dependent on physical conditions and emotional state. Since then, usually only meso-scale geometry deformations such as wrinkles or pores were considered, mostly in the context of aging processes (for instance [234]), but also concerning emotions, like in the MARC system presented by Courgeon et al. [47], which also includes an automatic wrinkles calculation. A real-time method for animating changes in skin coloration and droplet flow in the case of tears and sweat based on pre-defined key-frames encoded in a 3D texture was recently presented in [155].

Adding subtle changes in the facial color that relate to mimic skin distortions can help improve realism, although, as was shown by MacDorman et al. [208], the more texture photorealism and polygon count increased, the more mismatches in texturing etc. resulted in making a character eerie. Thereby, in the study of Pan et al. [238] on human participants' reactions towards blushing avatars, one of the findings was that people noticed the avatar's cheek blushing more than whole-face blushing. But as an even more important outcome, the results suggested that people were less tolerant if only the cheeks were blushing. Obviously the latter was not convincing as a blushing response, and thus this type of blushing was worse than having no blushing at all.

But nevertheless, the study indicated a strong correlation between whole-face blushing and sympathy. Although the participants noticed the whole-face blushing less, they felt increased co-presence with a whole-face blushing avatar even though they have not been consciously aware of it. This corresponds with the findings described in [66] that a blush can remediate others' judgments after clumsy behavior. Likewise, the experimental study on emotion perception recently presented by de Melo and Gratch [57] suggested that considering physiological manifestations are especially useful to convey intensity of emotions.

In AI simulating emotions is an important topic, and due to its computability simulation is often based on the appraisal-based emotion model presented with the OCC theory [233]. Thus, with ALMA Gebhard [91] introduced a layered model of affect for enhancing simulated dialogs based on the OCC model, whereas emotions are calculated within the three-dimensional, continuous PAD space (i.e. pleasure, arousal, and dominance). In this regard, affect influences the character's mind. Based on the particular moods and emotions, dialog strategies and communicative behaviors are chosen. In his model three types of affect based on different temporal characteristics are distinguished: first emotions, to which facial expressions belong and which are short-term affects; then moods; and

finally personality, which specifies a character's general behavior.

As was explained in Klesen and Gebhard [181], the affective state is then used to control gestures and mimics or even facial complexions in real-time (the latter being realized following [155]). Whereas in [181] emotions are used to control facial expressions, skin tone changes, and other affective animations like weeping, moods are mainly reflected by postures. Therefore, these are used to control the posture and idle behavior (e.g. breath or eye blink rate). Exuberant characters for instance show more body and head movements than bored ones, who might only look at their virtual watch from time to time.

In their WASABI affect simulation architecture, Becker and Wachsmuth [21] further differentiate between primary and secondary emotions in order to account for cognitively more elaborated emotions. Other relevant models, including the influence of personality traits and social circumstances, are also discussed in [319]. Also in robotics, emotional aspects like mimics, and very recently even vascular expressions,[8] are considered, which may indicate that this kind of topic will be developed further in the future.

Obviously the aforementioned emotional reactions only happen in the case of very strong emotions. For industrial and similar applications like e.g. an online manual they are probably not relevant, but there are others like an edutainment application that targets at affective educational objectives where visualizing strong emotions can be very important. In addition, if the skin tone is consistent with emotive expressions, this helps resolving ambiguities on perception. Thus, in the following chapters a framework based on current standards for visualization is outlined, that deals with the mentioned issues in order to simplify the integration of virtual characters.

[8] An image of a blushing toy robot for instance can be found here:
http://www.spiegel.de/netzwelt/web/0,1518,grossbild-1315143-581421,00.html

4 Character Dynamics

This chapter describes algorithmic building blocks and scene-graph nodes that are necessary to fulfill the requests from the PML-based control layer presented in chapter 7.2, which is responsible for specifying and synchronizing animations and related events. Because this requires to have the accordant features on the X3D level, we propose a set of self-contained nodes for realizing these demands. This includes for instance an audio node for text-to-speech that automatically calculates the phonemes and weighting factors for the corresponding visemes to achieve lip synchronization as well as dynamic gestures and techniques to generate animations during run-time [183, 156, 145, 147, 149, 160].

These components are part of the execution layer [147, 149] and are integrated into the X3D-based Instant Reality framework [135]. Following our distinction between consciously controlled and unconsciously happening actions described in section 4.2 [149], this chapter is split up into two parts. After discussing consciously controlled behavior and character animation in general as main requirement for multi-modal dialog systems, a robust and efficient method for real-time hair simulation and rendering is described in section 4.5, and after that emotional aspects and other "unconsciously" happening phenomena are discussed in the following chapters.

4.1 Character Setup

First, we focus on defining and controlling humanoid animation in the context of X3D [336]. The open ISO standard X3D provides a portable format and runtime for developing interactive 3D applications. It is the only ISO-certified and royalty-free open standard for 3D interchange file formats and runtime architectures. Thereby it is one of the most widely used standards for the implementation of high-integrity 3D systems, because as a direct evolution of VRML models that are over 15 years old still run today. The humanoid animation component (H-Anim) [335] now is also part of the X3D standard, and it not

Figure 4.1: *The verbal and non-verbal communication of this discussion is controlled with our proposed animation controlling component [147].*

only provides a well defined skeleton structure of humanoid figures but also support for character animation based on predefined keyframe data.

In addition, X3D allows defining scene description and runtime behavior by simply editing text or XML files without the need for dealing with low-level C/C++ graphics APIs, since it is a high-level language on top of OpenGL or DirectX. X3D's main advantage is bringing 3D graphics to a wider audience that is not necessarily skilled in 3D graphics programming but typically writes the behavior part of a 3D world. With X3D these people can create meaningful representations of their data without specialized tools or low-level graphics knowledge, which is of great importance for efficient application development.

4.1.1 The X3D/ H-Anim Standard

The original H-Anim standard only defined the skeleton setup, consisting of the rigid segments and joints that are needed to build up a humanoid. Different levels of quality (levels of articulation, LOA) are defined. As can be seen in Figure 2.1, this joint hierarchy thereby extends the scene-graph. Later a simple but efficient skins and bones refinement of this ISO standard also introduced seamless skinning. Additionally displacer nodes were introduced for modifying the mesh with a set of displacement vectors.

Standard VRML (or later X3D) animation techniques, mainly timers and simple linear interpolators, have been used to change e.g. the joint rotations over time based on predefined animation sets. The animation data itself is stored in X3D *Interpolator* nodes; up to four interpolators per joint, and the data flow is defined via X3D routes. However, the definition and handling of dynamic changes have never been part of the H-Anim standard that only defines the structure, and the built-in X3D animation mechanisms are not suitable for dealing with multiple animations that shall be played-back concurrently. The following X3D example in VRML encoding shows the animation of one single joint.

```
DEF HUMANOID HAnimHumanoid {
    ...
    DEF HANIM_R_HIP HAnimJoint {
      name "r_hip"
      skinCoordIndex  [...]
      skinCoordWeight [...]
      children [ ... ]
    }
    ...
}

DEF TIMER TimeSensor {}
DEF R_HIP_ANIM OrientationInterpolator {...}

ROUTE TIMER.fraction_changed   TO R_HIP_ANIM.set_fraction
ROUTE R_HIP_ANIM.value_changed TO HANIM_R_HIP.set_rotation
```

For simple scenarios, like a single animation to be played, H-Anim/X3D works well, but it is hard to use in cases where multiple animation sets are available, which shall be combined

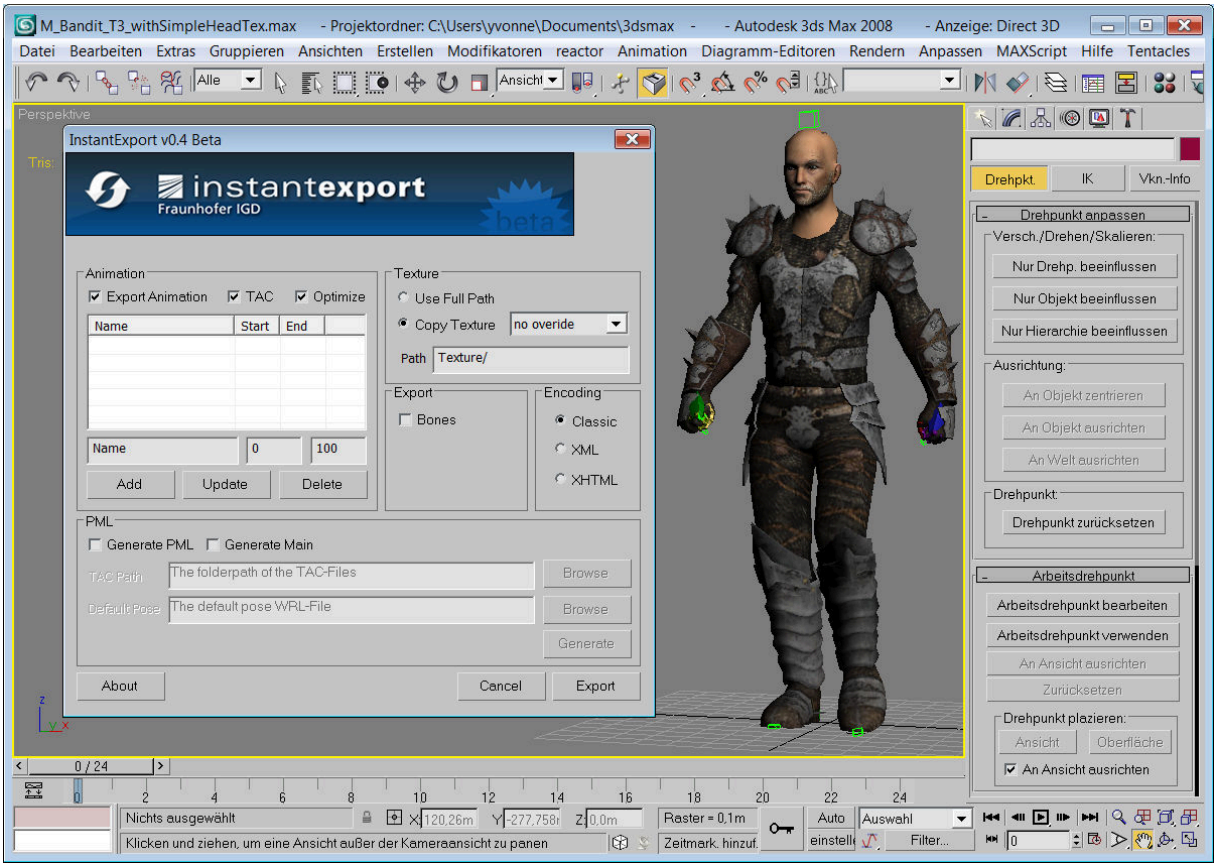

Figure 4.2: *Screenshot of our X3D/ H-Anim exporter for 3ds Max [12]. Using open standards for creation and exchange of assets is important for more sustainable solutions.*

and concatenated dynamically during run-time as visualized in Figure 4.3. There is neither built-in support for cross-fading and transitions between motions nor for motion blending. Hence, for most use cases doing key-frame animations this way is too fine-grained and obviously does not support animation synchronization etc. The overall structure of such an application also gets unmanageable and confusing because of increasing complexity due to the vast amount of nodes, routes and missing information about membership to specific animations. Tracing and debugging is almost impossible, especially when routes are created and deleted during run-time to blend animations together in *Script* nodes.

However, for realistic scenarios this gets even more complex, because the various types of dynamics concerning humanoids is a very important aspect and occurs in different ways. The most obvious are the character's movements, like gestures and locomotion. But also hair is not static and must be simulated to achieve high visual realism. Generally spoken, two types of approaches can be distinguished: the play-back of predefined animation data on the one hand, and the on-line computation of animation data (e.g. via inverse kinematics, IK) on the other hand. Suitable techniques are discussed in section 4.3.

In addition, in the context of multi-modal dialog systems a high-level command language, which allows the description of actions on an abstract level like "go to red box" is necessary for hiding pure graphics commands within the animation engine. To fulfill such abstract commands in real-time, including blending of commands that are acting on the same

elements in the scene, flexible animation capabilities are required. Furthermore, X3D does not provide any support for more advanced features like text-to-speech (TTS), including the automatic calculation of phonemes and weighting factors for achieving lip-sync.

Besides this, the current X3D standard still lacks advanced rendering techniques like shadows, multi-pass techniques, and fine-grained render state control, which not only are essential for realistic rendering but are also actually state of the art. The same goes for physics. Although since the latest specification revision a rigid body physics component is part of the standard, there is still no support for deformable objects. Summarizing it can be stated that the existing X3D concepts provide a good basis but are not sufficient. Thus, suitable extensions are discussed in chapters 5 and 6.

4.1.2 Data Exchange Issues

To create a virtual character, first its visual appearance must be modeled using a 3D modeling package like 3ds Max [12] or Maya [14]. This comprises the creation of the polygon mesh as well as applying textures for high visual quality. It needs time and talented designers to create good character models [183] and is out-of-scope here.

Although there are lots of modeling tools, animation engines, and data formats for describing virtual characters and animations available (e.g. OBJ, BVH, FBX, Collada [10], MPEG-4 [239], and X3D/ H-Anim [336, 335]), most of them are proprietary (e.g. FBX), to some extent unusual (like MPEG-4), or they do not define any kind of runtime behavior (all but the latter two). Another issue is that file formats like FBX or MAX are constantly changing with every new software release and usually not backwards compatible.

Because X3D is an open ISO standard, scene, character and animation data is portable, which is essential for content creation. Further, by utilizing X3D we want to achieve a more sustainable solution by building upon standardized base elements instead of proprietary tools [149]. Many of the common 3D modeling tools like Cinema4D [216], Blender, Maya and 3ds Max already provide an X3D exporter, but in many cases it is currently only possible to export a certain subset of X3D, namely the old VRML standard.

Besides Maya [14], 3ds Max [12] is one of the most common tools here, but it still only supports X3D's predecessor VRML. Thus, we have implemented an exporter plugin (Figure 4.2) that exports skinned characters as H-Anim as well as all animation data either as standard X3D or in our proposed container format as shown in Figure 4.5.

Another tool, which seemed to us as fairly straightforward to use, is Poser.[1] The software offers characters ranging from comic-style to realistic, and adaptations of the characters can be carried out, too. Most character models provide a wide range of body animations and morph targets, which can be simply applied to form more complex animations. Even animating a character by hand is straightforward and allows creating animations in a short time. Unfortunately, Poser's export options are very limited and erroneous, why we also developed an exporter for Poser based on its Python API in order to directly export characters and animations into H-Anim/ X3D compliant files, by also supporting the additional features presented in the next sections.

[1] http://poser.smithmicro.com/poser.html

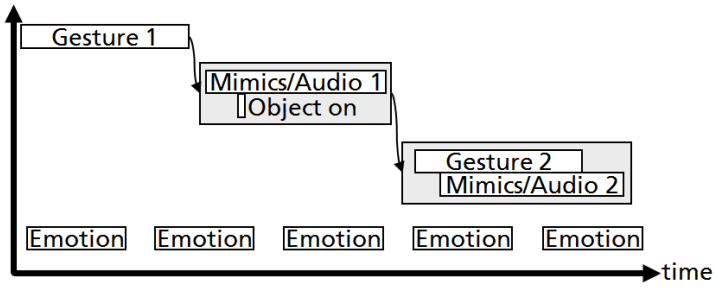

Figure 4.3: *Timeline exemplarily showing succeeding and concurrently happening actions.*

4.2 Building Blocks and Execution Layer

Not only in the field of X3D but also in general, animating and rendering virtual characters still has a lot of challenges. First, the methods should be easy to use and integrate into different applications such as assistance systems etc. Second, to have a flexible control of the character requires a flexible animation system including body movements (e.g. gestures, walking) and speech (text-to-speech, mimics). Then, visual realism means to have realistic models, natural gestures and a realistic simulation of materials.

Ultimately, for interactive systems, everything has to be done in real-time, because in typical applications, like game scenarios and embodied conversational agents for entertainment and education, the interface must react immediately to user input, usually by combining different gestures and postures. But currently for instance neither the exchange format Collada nor X3D support such advanced interaction and animation methods.

Moreover, if a virtual character also shall speak, the situation is even worse. Because, as already mentioned, X3D does not provide support for text-to-speech, all spoken texts have to be prepared in advance, and – after having determined the lengths of all phonemes – put in synchronization with the corresponding visemes somehow. But in general, this is quite frequently handled in an extra module, as the timing information for speech needs to be known beforehand if the corresponding gestures, which moreover depend on the modeled personality, shall be temporally aligned with the verbal utterance.

4.2.1 Consciously Controlled Behavior

When virtual characters represent the dialog interface and act as personal dialog partners, a reliable and consistent motion and dialog behavior is essential. Here, dialog behavior not only embraces story, dialog management, speech and gestures, but also affective aspects. But as was mentioned in the introduction, in the area of computer graphics and multimodal dialog systems research normally focuses on face and body animations, following the typical 3-stage model of human information processing (sensory perception, decision/cognition, motor response), thereby only considering body functions controlled by the central nervous system like voice and motor response.

In contrast to the central nervous system, which is responsible for the conscious control of motor functions, the autonomic nervous system controls unconscious inner functions that

can result in physical reactions like blushing, pallor and tears. However, issues concerning rendering and simulation are mostly ignored, although increasingly powerful computers allow for a more realistic character and scene design. Hence, section 4.2.2 shortly outlines our approach to overcome this problem. More information will be given in chapter 5.

However, since gestures and speech are the most important factor for our use case, the next few sections mainly present basic animation techniques and how they can be integrated into X3D to enable advanced animation control. In order to combine and concatenate animations efficiently, as e.g. needed for simulating the communication between different virtual characters and a user as shown in Figure 1.3, we propose using animation storage nodes that provide a consistent view on a given animation set. Furthermore, to provide a consistent and transparent interface for the application developer we have designed a centralized control component for animations and related actions [156, 147, 149].

Hence, in chapter 7 we propose a layered approach for achieving advanced animation control by splitting up the complexity into different layers in order to overcome the outlined problems inherent in X3D. As can be seen in Figures 1.4 (on page 28) and 7.1 (on page 204), there is a hierarchy of different levels of abstraction concerning modeling virtual characters. Although hierarchical classification is a traditional approach in character animation [98], we utilize current X3D nodes and concepts [147].

Here, the lower X3D-based levels make up the execution layer (shown in green in Figure 7.1), which deals with geometry, appearance, and dynamics. Somewhere in between these layers is physics, ranging from rigid body physics over hair and cloth simulation to the simulation of complex materials like skin. Moreover, for being able to also account for physiological effects and the like, we further distinguish between consciously controlled behavior and unconsciously happening phenomena. Corresponding techniques including their X3D integration are discussed in the next chapters.

In contrast to this, the control layer (in Figure 7.1 shown in amber and beige respectively) is responsible for handling instinctive behavior and cognition and is discussed in more detail in chapter 7. As exemplarily visualized in Figure 4.3, the proposed control layer coordinates and synchronizes actions in time. It is not directly part of X3D but builds on top of it by introducing the animation scripting and representation language PML.

By using these X3D extensions for simplified behavior control that are further explained in chapter 7, not only scripting but also mixing of animations, provided that they are given as rigid body motions, can be easily done in X3D as shown in Figure 4.4, since all necessary data and program logic is handled internally by the X3D player, yet allowing the application developer to concentrate on the content.

4.2.2 Unconsciously Happening Phenomena

At the execution level we further distinguish between processes taking place consciously and which need to be animated explicitly vs. phenomena that do happen unconsciously and which cannot be controlled intentionally, like the motion of hair when moving the head, tears running down a face (as e.g. shown in the middle of Figure 5.13 on page 153), or even shadows, but which also have to be simulated for creating plausible characters. Regarding phenomena that happen without any intentional control, again basically two

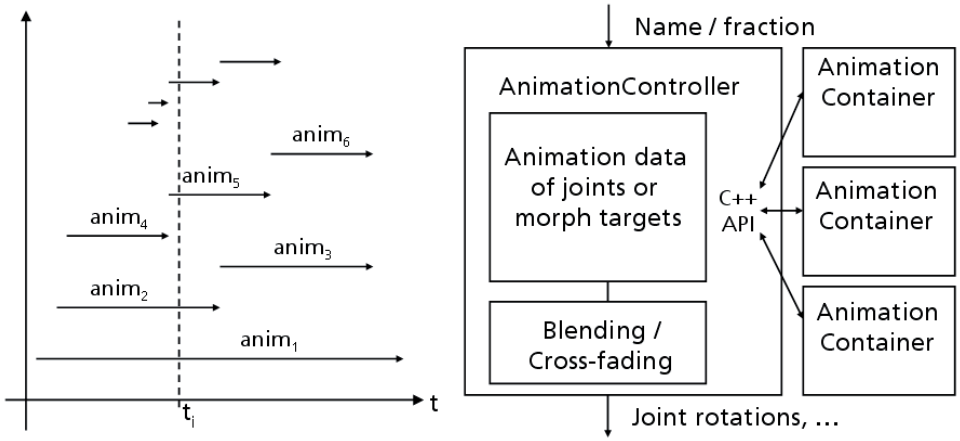

Figure 4.4: *Timeline at time t_i (cp. Figure 4.3), and mixing in AnimationController.*

cases can be distinguished: some phenomena like crying or blushing are part of subconscious behavior and need to be triggered explicitly by the behavior controller for being synchronous to other animations, other effects such as hair motions simply have to follow.

In this last category all dynamic effects that are mostly not specific for virtual humans but of interest for all scene elements, like lighting and shadows or the motions of deformable objects like grass, hair and cloth, have to be covered [149]. Thus, we have also implemented nodes for doing hair simulation and rendering, which are described in section 4.5. The simulation is based on a kinematic multi-body chain, whose nodes are defined by the vertices of the original hair mesh that consists of many quad strips.

Such adjoint effects and resultant dependencies are not managed by the animation control system but are directly integrated into the scene-graph. This is described in chapters 5 and 6. Whereas the dynamics of skin and hair can only be triggered indirectly (e.g. by changing the parent transform), other events, like the crying simulation explained in section 5.3, need to be controllable explicitly. As demonstrated in section 7.2.3, for higher level control here the same concepts as for standard animations apply.

4.3 Body Animation

4.3.1 Playback of Predefined Animations

As mentioned, the X3D H-Anim component [335] provides support for character animation based on predefined animations, which can result from motion capture data or hand animated motion data. H-Anim is based on a Skins and Bones model and defines a common skeletal model including locations and names of specific bones. The animation data itself is stored in X3D interpolators, and the data flow is defined via X3D routes. For efficient combination and concatenation of animations (some examples are shown in Figures 4.1 and 4.6) additional information about the animations is also needed, like data look ahead and a list of active animations and animations that will be activated within the next time-frame, which the humanoid animation component does not provide.

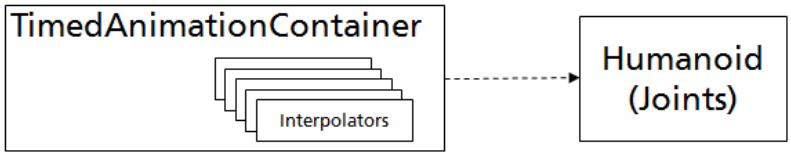

Figure 4.5: *In contrast to the standard X3D approach as depicted in Figure 2.2, an AnimationContainer encapsulates all data that is specific for a single event or animation.*

To alleviate this drawback, we have designed animation storage nodes that primarily act as data containers and which provide a consistent view on an animation set, including membership information of nodes to a specific animation [183, 156]. Figure 4.5 visualizes the proposed character data organization. The animation container node is designed as a container for all animation data like the X3D *Position-* and *OrientationInterpolator* nodes as well as the target joints of one specific character animation (e.g. wave hand).

The storage nodes inherit from our new *X3DAnimationContainer* that contains the animated targets of an animation and whose node interface is shown below (Figure 4.5). The MFString field 'targetnames' holds references of targets to be animated or changed. These usually consist of the joints (which are specialized *Transform* nodes) – but in case of e.g. blushing it can also reference a texture transform or a shader program.

The MFString field 'fieldnames' contains the names of the corresponding fields in order to find this field inside the target. This is needed, because if for instance an SFVec3f value shall be sent to a target node, e.g. a *Transform* node, it is often ambiguous, which field was meant (in this example it could be either of 'center', 'scale', or 'translation', which are all of type SFVec3f). Moreover, when dealing with script or shader nodes, e.g. uniform shader variables are integrated as dynamic fields and thereby known not until instantiation of the concrete node.

```
X3DAnimationContainer : X3DAnimationBase {
  SFString [] name        ""
  MFString [] targetnames []
  MFString [] fieldnames  []
}
```

As shown in the example timeline in Figure 4.3, it is often necessary to playback multiple animations like waving and turning around at one single time step. In addition, an emotion module can also send further animation requests at the same time. Thus, for doing convincing character animation in complex and responsive environments, H-Anim [335] needs to be further extended to incorporate blending of different animations, which cannot be accomplished with current X3D concepts [147, 149]. The same goes for cross-fading different succeeding animations in order to alleviate jerky leaps between e.g. an idle motion [75] and a subsequent gesture. This gets even more complicated, if motions from a physics simulation or an IK system, which besides this both need some knowledge of the given scene, also shall be incorporated.

Therefore, we also developed a centralized control engine for animations and related actions to provide a consistent and transparent interface and to overcome the problems

Figure 4.6: *Mixing of two motions (end positions of both arm movements marked in red).*

mentioned above. The concept basically is visualized in Figures 4.4 (right) and 7.8 (page 218) and is explained in detail later in section 7.3.4. The proposed controller component additionally provides an interface for scripting and scheduling different types of actions and animations for encapsulating simple behavior, by means of the *TimelineComposer* node that acts as the interpreter of our animation scripting language PML [147, 149].

Alternatively one could think of defining the temporal order and triggering the corresponding actions by implementing a time graph structure consisting of parallel and sequential *TimeContainer* nodes, as was shortly discussed in section 2.1, for mapping the animation time to the fraction of the final key-frame intervals. But for more complex schedules this soon gets confusing and hardly maintainable. Furthermore, it does not provide any means for defining animation data centrally and mixing it with other animations.

Hence, triggered per frame by the scheduler, as will be explained in section 7.3.4 (cp. Figure 7.9 on page 219), and based on the animation's names and current fraction the *AnimationController* node first fetches all relevant animation information of a given object for time t_i (see Figure 4.4). Then it blends and cross-fades the animation data in case more than one animation is active at the same time, and finally it writes back the new transformation values into the according fields of the joint nodes. An intricate application setup is thereby avoided and the treatment of different animation types is unified.

Our cross-fading mechanism dynamically blends between succeeding animations. Depending on how much the temporally succeeding rotations differ from each others (which can be further parameterized in the animation controller's node interface described in section 7.3.4.2), a substitute rotation is computed, until both rotations converge. Thereby jumps between animations are avoided and a temporarily dynamic cross-fading phase is attained. The additional benefit here is that artificial helper structures like blend trees [72] can be avoided for blending gestures. In the last step all quaternions are multiplied with their weights, combined and finally normalized for obtaining the interpolation result.

By using this animation controller extension (see chapter 7.3.4), which is an abstraction

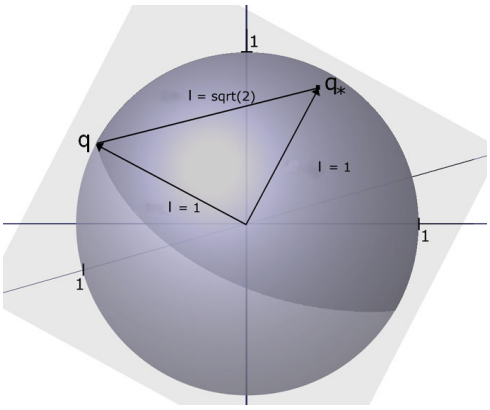

Figure 4.7: *4D unit hemisphere (simplified to $\mathbb{R}^3$) with reference quaternion q_*.*

for all sources of motion data, mixing of animations can be easily done in X3D and thus reduces the complexity of application development. The principle is conceptually the same for facial animation based on morph targets [4] or H-Anim displacer nodes (compare section 4.4), or for animating colors in the case of blushing. Figure 4.6 shows an example, where the position of the character's right arm was obtained by blending two different animations, whereas the red boxes symbolize the respective end positions of the hand for both animations without blending them [149].

But there are still issues, that have to be kept in mind when mixing animations, mainly due to unsuitable animation data and missing transition animations. Further, rotations internally are represented as unit quaternions. Because the same rotation can be described by the quaternions q and $-q$, care must be taken when blending quaternions. In order to get a unique description of a rotation, we first define a 4D unit hemisphere, on whose surface S the unit quaternions are located. Following [242], the initial choice of the hemisphere is based on an arbitrarily chosen reference quaternion (e.g. the first one, depicted as q_* in Figure 4.7).

Our simplified method [145] is based on the observation that the reference quaternion q_*, any other quaternion q, and the origin of the 4D unit sphere are always coplanar. With the additional constraint, that all unit quaternions are located on the same hemisphere S, the maximum angular separation between q and q_* is 90°. The maximum distance between two unit quaternions is $\sqrt{2}$ and directly results from Pythagoras. By calculating the Euclidean distance d between q_* and any other quaternion q, we can check, if q lies on the reference hemisphere by comparing d with $\sqrt{2}$. If $d > \sqrt{2}$, then q does not lie on S. By simply negating q the given rotation is represented by a quaternion located on S.

4.3.2 Locomotion Generation

While communicative gestures can be easily blended with the aforementioned approach, there also exist cases where more or less severe artifacts occur. If e.g. in an animation loop the spatial distances between the first and the last animation frame are too big, this either leads to jerks or to sliding effects, depending on the blending parameters, which in the latter case introduce damping effects, if too many time steps are averaged. Thus,

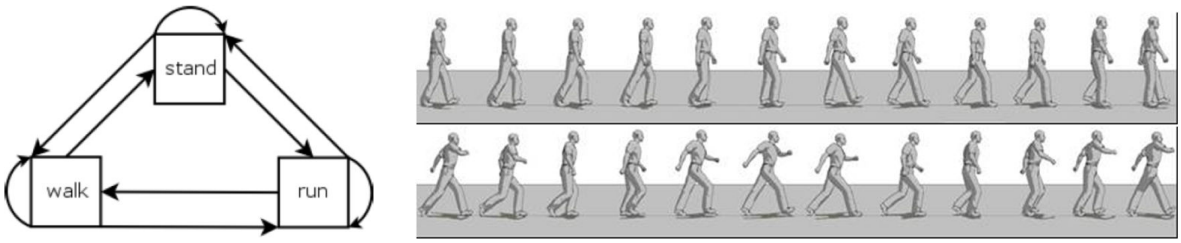

Figure 4.8: *Left: example of a simple motion graph [241]. Right: different walking styles.*

pre-recorded animation must be planned accurately, and it should be defined, which is the starting and which is the ending pose, as well as which joints are involved.

Because blending between very different poses often leads to unsatisfactory results, it thus should be avoided. If for instance the animations are too different, blending can lead to artifacts like foot sliding or even interpenetrations of extremities. Instead, transition motions can be used, e.g. to plausibly fill the gap between a sitting and a walking animation. Nevertheless, not all animations can be planned in advance and as outlined in section 2.1.3.2, motions often need to be dynamically generated or parameterized. For containment, in this section we will focus exemplarily on locomotion, i.e. walking.

Basically there exist two types of approaches for automatic generation of walking animations [222]. The first one is model-driven and tries to simulate the physiology of the human body using kinematics and dynamic constraints. The flexibility is very high, because theoretically any kind of human motion can be calculated, but for decent results the complexity of such simulations is very high, too, thus real-time calculation is hard to achieve, if at all. While e.g. gaze behavior is rather simple to synthesize at runtime, for other motions such as walking (like for "go to door") this remains an under-determined, computationally intense problem that often results in unnatural, robot-like animations and additionally requires semantic knowledge.

The second type are data-driven methods that adapt captured motion data according to external parameters such as motion path and velocity (cf. section 2.1.3.2), e.g. "interpolation" between walking and running to attain jogging. Here the complexity concerning biomechanical constraints is much lower, because originalities of human walking are already defined in the animation sets. But the drawbacks of these approaches are retargeting issues [99] and the need for a lot of motion data upfront [192].

The most promising approach we found belongs to the second type and was the one described by Park et al. [242, 241]. It synthesis animation data from previously captured animation data according to different parameters, e.g. mood of character and style of walking (see Figure 4.8, right). In a first step one has to pre-process the motion data based on the transition graph (see Figure 4.8, left) and create animation sequences for the transition motions. Each sequence comprises of one walking cycle with fixed speed, angle and mood. To walk on a given path or towards a specified target these sequences are automatically concatenated during runtime. The values itself are then inter- or extrapolated according to the input values defined by the application.

We created the necessary motion data for testing with a specially written Poser exporter since no MoCap data was available. The calculation of the weights for the animation data

– 56 joints, 3 parameter dimensions (path, velocity, and style), 27 example motions (all 3^3 combinations of the parameter dimensions) – took approximately 26 to 36 ms on a Pentium 4 with 2.4 GHz and is thus real-time capable. The visual results were very convincing and especially the concatenated walking cycles looked quite life-like [156, 145]. A virtual character can thereby dynamically follow a given path without foot sliding artifacts that occur when ignoring the direction of the motion path. In addition, as opposed to Park et al. the motion not only can be parameterized based on velocity but also on style (e.g. walking, jogging, running) and emotional state like sad, neutral, and happy.

The algorithm was integrated into our framework by introducing two specialized subclasses of the proposed *AnimationContainer* for handling locomotion generation. The example motions are hold in *LocomotionAnimationContainer* nodes that inherit from the *TimedAnimationContainer* presented in section 7.3.4 by extending it with fields for parameter values and generic time keys that denote when the foots touch the floor and the arms are up or down in normalized time for dynamic time warping. These container nodes are referenced by the *BlendingLocomotionContainer* that does the preprocessing for the generated locomotions such as the calculation of weighting functions during initialization based on estimated coefficients for the blending function. It also has an MFFloat slot "paramVec" that holds the current values of the given parameter dimensions.

4.3.3 Dynamic Gestures

4.3.3.1 Inverse Kinematics

Capturing and processing motion data is a tedious, expensive, and time consuming task. In addition, data driven approaches require lots of example motions that often are not available and do not allow for goal-directed motions. To increase flexibility sometimes a better solution is to automatically generate animation data. Furthermore, there are animations whose appearance is not known upfront because they depend on external parameters. Hence, another issue is the generation of goal-directed motions that usually is achieved with procedural methods like Inverse Kinematics (IK). Examples are pointing gestures, where the direction is calculated on-the-fly (e.g. pointing towards a moving object), and gaze behavior, where the target is defined during runtime ("look at Tom").

As can be seen, such animations need information about the scene: "go to A", "look at B" etc. implies having knowledge of the avatar configuration and of A or B. By using external modules for the computation (the only possibility in X3D), in dynamic worlds this denotes high latency and unnecessary communication overhead, especially when taking animated terrain or other moving targets into account, and in static scenes it means at least keeping data twice, like for instance for path planning. Therefore built-in support for inverse kinematics would help to alleviate these drawbacks.

But currently the only built-in way of animating humanoids within X3D is via keyframing, although the H-Anim component already contains a node, the *HAnimSite*, that can be used to generate another type of animation, and which generally can serve three purposes [335]: It defines an 'end effector' location, which can be used by an inverse kinematics system (but without specifying the animation itself and how it could be triggered), an

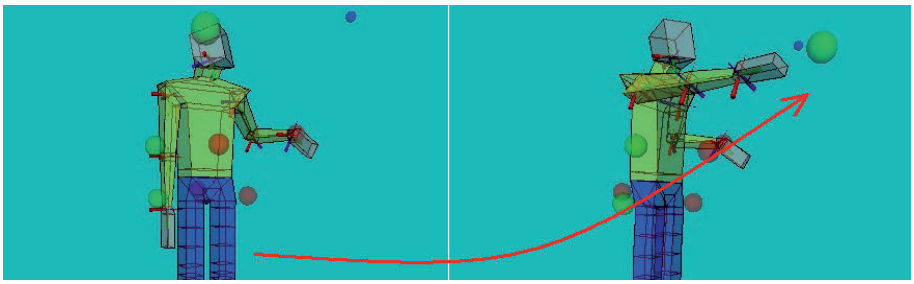

Figure 4.9: *The "boxman" is aiming at the big green sphere with his right arm via the HAnimIKSite node – the resulting motion is denoted by the red arrow.*

attachment point for accessories, and a location for a virtual camera.

Only defining the position of an end effector in the given reference frame usually is not enough for fully parameterizing an IK system, and also doesn't provide enough information about the targets of the motion to be calculated. For example a simple 'lookAt' gesture requires knowledge of the target at which to look at, but that target (e.g. the eyes of another character) in general is not part of the humanoid. Therefore we propose the *HAnimIKSite* node [147], whose implementation is based on the algorithm presented in [129], for explicitly handling inverse kinematics within X3D. Its interface is shown below (fields already defined in the *HAnimSite* node [335, 336] are omitted for clarity). Since different *HAnimIKSite* nodes may act on the same joints, for each character the final motion values are controlled and merged in the *HAnimHumanoid* node.

```
HAnimIKSite : HAnimSite {
  [...]
  SFFloat    [in, out] fraction          0
  SFVec3f    [in, out] target            0 0 0
  SFRotation [in, out] targetRotation    0 0 0 1
  SFVec3f    [in, out] aimingTranslation 0 0 0
  SFRotation [in, out] aimingRotation    0 0 0 1
  SFInt32    [in, out] numJoints         3
  SFFloat    [in, out] motionPathTension -0.4
  SFString   [in, out] motionPathShape   "auto"
  SFFloat    [in, out] minTurnAngle      0.2
  SFFloat    [in, out] maxTurnAngle      1.5
  SFFloat    [in, out] turnFactor        0.6
  MFString   [in, out] jointNames        []
}
```

The SFFloat field 'fraction' can be set (e.g. by a standard *TimeSensor*) to values between 0 (start) and 1 (end) to animate the avatar. The 'target' field defines the position of the target to aim at or to touch in world coordinates. Usually the translation of a *Transform* node is taken here (symbolized by the green sphere in Figure 4.9). If the target should not be aimed with the center of the parent joint of the *HAnimIKSite* (maybe to define a point lying between the eyes or in the hand), then a translation/ rotation of the end effector can be defined via the 'aimingTranslation'/ 'aimingRotation' field. The length

of the kinematic chain is defined by the 'numJoints' field (albeit currently only one- and three-joint kinematic chains are possible in our implementation).

The value of the 'motionPathTension' field has to be in $[-1, 1]$, and defines the tension of the motion path in the sense of the Kochanek-Bartels spline tension parameter [109, p. 59]. The 'motionPathShape' field determines the shape of the motion path, currently valid values can be "auto", "quadratic", and "cubic". The 'minTurnAngle' defines the minimal angle a target has to be away in order that the torso turns, 'maxTurnAngle' defines the maximal angle the torso can turn, and 'turnFactor' is the factor, by which the turn angle is decreased. Finally, the MFString field 'jointNames' contains the names of the joints involved within the kinematik chain, e.g. one name for the "vl5" joint (referring to the joint names as standardized in the H-Anim specification).

4.3.3.2 Motion Segments

Although basically IK allows doing any kind of motion, the computation can get arbitrarily complex and time consuming for animations like grasping or walking due to the high number of degrees of freedom. Also, IK not only assumes that the character fulfills certain requirements according to its joint configuration and limb setup that mostly cannot be ensured when using a standard 3D modeling tool like 3ds Max (such as an initial T-pose or the H-Anim humanoid default position with one of the supported levels of articulation and identity matrices for all joints). Moreover, for correct animations it is also necessary to rely on biomechanical constraints like the rotation limits of a shoulder that cannot be rotated more than x degrees around a given axis etc.

These are likewise not exposed by a typical character model, which in general is nothing but a mesh with some bones attached to it, while muscles, tendons, and the like are more or less ignored here. To alleviate these problems in case no suitable character model is available even for realizing simple motions like "point at", and because it is costly to create all kinds of animations in advance (or at least enough example motions as necessary for the method described in section 4.3.2), we follow the so-called channel concept, which is also utilized for instance in [123]. The basic idea is to divide the body into certain parts, where each part of the skeleton can be animated by appropriate motion segments.

To be able to specify or request such motions dynamically during runtime, these motion segments, which only affect certain body parts like the head or arm, can then be controlled and combined via a higher-level rule engine and PML [160]. As will be explained in more detail in section 7.3.4, all animations including the motion segments then are sent to the internal scheduler. After that, for the given time frame t all active animations are merged and blended as described previously in section 4.3.1. E.g. by mixing a point-down with a point-up gesture an in-between position can be achieved (compare Figure 4.6), whilst it still remains possible to play-back another animation at the same time.

4.3.4 Motion Synthesis and Planning

To efficiently integrate synthesized and goal-directed motions into our framework, we follow the successive approach described in [302] (at least on a basic level), which is based

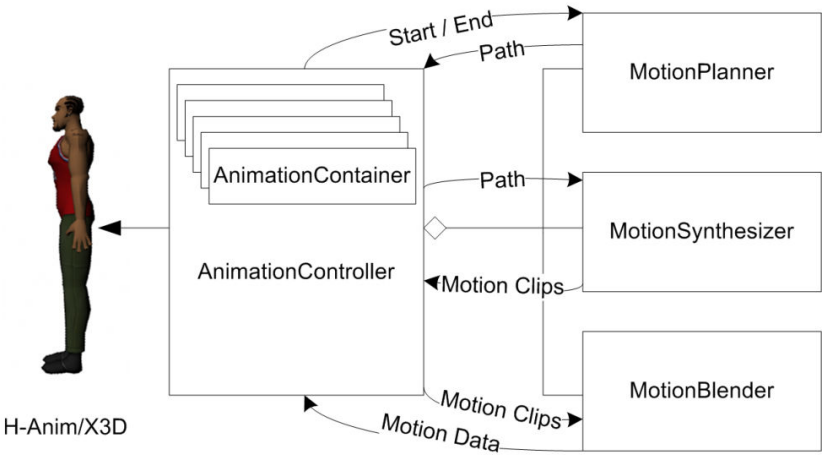

Figure 4.10: *Module stages building on top of each other for motion planning. These stages are controlled for each character by the AnimationController node. Note that motion synthesis covers different techniques like data-driven and procedural animation, and thus denotes the methods that are used to generate the animation data.*

on the motion-graph technique presented in [192]. Although the presented motion-graph approach is not suitable for real-time use, the basic principles can also be applied to interactive applications [149]. The idea is to first coarsely plan the path. With this information motion clips are then composited, which in the last stage are interpolated for smooth results. Suppose the following little PML animation script (whose use and syntax will be explained in chapter 7) shall be executed:

```
<moveTo refId='fragment' pos='t' orientation='R'>
```

Here, the additional PML animation element `<moveTo>` can be used to define a path along which an object or a camera can be moved. This path can be derived from a floor plan or by querying the scene taking possible obstacles into account. Thereto, first the path planning module has to be fed with the start pose $R_0|t_0$ and target pose $R|t$. After that, the calculated path $< R_0|t_0, ..., R_i|t_i, ..., R|t >$ is propagated to the motion synthesizer. And finally, the returned motion data is used as input for the motion blending module. In Figure 4.10 the main concepts are depicted [149].

So first, the motion path has to be planned, e.g. for walking or grasping, and after that, the motion has to be synthesized, e.g. via inverse kinematics or a graph walk. Internally this can also be realized by creating an *AnimationContainer* (Figure 4.5) with corresponding interpolators. In a final step, the calculated motions have to be blended for a smooth result (Figure 4.4). The advantage here is, that depending on the application several steps can be skipped: if for instance the character shall only gesticulate during the conversation, the motion synthesis step can be reduced to retrieve a pre-defined animation clip, and if there is only one animation at a time, there is no need for further blending.

In Figure 4.6 another case is shown, where no path is needed, but different animations at a time need to be played-back and thus mixed, similar to the method outlined in the last paragraph of the previous section. The screenshot shows the basic idea of the blending mechanism, whereas the screenshots in Figure 4.1 or Figure 1.3 on page 27 both show a concrete scenario with talking characters. Thereby, our proposed approach is scalable

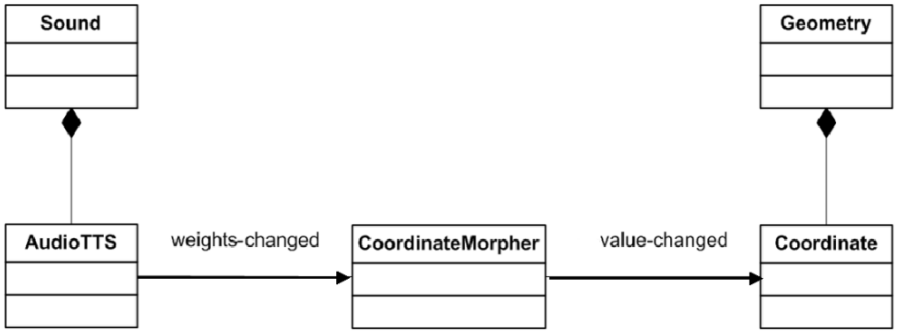

Figure 4.11: *Online TTS with output of weights and "VisemeKey" for animating the lips.*

and this way also accounts for a quality–speed trade-off.

4.4 Mimics and Speech

4.4.1 Speech Synthesis

Depending on the application different behaviors are required for animating virtual humans. Not only body motions and mimics but also speech and therewith some sort of lip synchronization is crucial for ECAs, but not yet supported by X3D. All functionality or high-level requests like `<speak>` respectively that is exposed by the control layer (via PML, cf. section 7.2), needs to have matching techniques in the execution layer (i.e. on the X3D level). When using e.g. conversational agents, interpersonal communication and therefore facial expressions, speech (preferably via online text-to-speech), and at least some sort of lip synchronization are crucial.

In X3D based applications speech synthesis usually is done externally and the facial animation is updated and controlled via Java and the EAI. When using the X3D *AudioClip* node it is certainly also possible to generate animated speech by preparing all spoken texts in advance or via some external modules. But if lip-sync is needed, first the lengths of all phonemes (usually provided by the text-to-speech system, too) have to be determined, before they can be synchronized with the corresponding face displacements.

This approach therefore not only leads to latency but also affords a lot of complexity concerning scripting and scene design. Hence, we propose an additional X3D *AudioTTS* node [147, 149], which is a text-to-speech (TTS) node that can be referenced by *Sound* nodes instead of an *AudioClip* node. The *AudioTTS* transforms written text to audio data by using a synthetic computer voice. The SFNode field 'voice' contains a *Voice* node, which describes the synthetic voice.

```
AudioTTS : X3DSoundSourceNode {
  SFString [in, out] text        ""
  SFNode   []        voice       NULL
  MFString []        visemeKey   []
  MFFloat  []        weightValue []
```

```
  MFFloat  [out]     weights_changed
  SFFloat  [in, out] visemeDurationScale 0.5
  SFInt32  [in, out] autoSilentIndex     -1
}
```

The *Voice* node defines further parameters of the computer generated voice. The 'name' field contains the name of the voice that corresponds to the voices, which are installed on the computer. If no name is given, the default voice is used. The 'gender' field determines the gender of the voice, possible values are "auto", "male" and "female", whereas the 'age' field determines the age of the voice, valid entries are "auto", "child", "teen", "adult" and "senior". The 'language' field finally controls the language of the voice (e.g. "en" for english or "de" for german; if not given the default language is used).

```
Voice : X3DSoundNode {
  SFString [] name     ""
  SFString [] gender   "auto"
  SFString [] age      "auto"
  SFString [] language ""
}
```

The SFString field 'text' of the *AudioTTS* node contains the text that gets spoken by that voice. Besides creating the audio data, this node can also provide weights to morph between different geometries (the morph targets). This can be used to animate the lips of avatars by mapping the resulting phonemes to their corresponding visemes (which describe a mouth movement). As shown in Figure 4.11, by providing viseme keys for the given morph targets the node additionally calculates weighting factors and phoneme lengths for animating the lips by morphing the face mesh correspondingly. Therefore the 'visemeKey' field is used: If for instance the viseme *a* is represented by the first geometry, an *a* is put at index 0 of this field, etc.

The concrete implementation of speech synthesis is platform dependent: for Microsoft Windows we use the very comprehensive Microsoft Speech API (SAPI 5.x)[2], whereas on Linux the Festival[3] library is used as there isn't much choice. Unfortunately, the C/C++ bindings of the latter are still rather rudimentary, for example neither the phonemes (which describe the smallest hearable units of a language) nor the phoneme durations can be retrieved correctly, thereby preventing the implementation of any useful lip-sync.

The 'weights_changed' outslot provides the weights for the different geometries used to create the final geometry. Usually, there is one geometry for each viseme provided by the text-to-speech system. For each geometry, one weight is calculated, whereas the sum of all weights is 1. After that, all geometries are multiplied with their respective weight and summed up. The result is an animation of the lips that is synchronous to the speech. The 'autoSilentIndex' is the index of the silent or neutral geometry. When this field is non-negative, the node ensures that the corresponding geometry is shown at the end of the animation sequence.

[2] http://msdn.microsoft.com/en-us/library/ee125077%28v=VS.85%29.aspx
[3] Festival Speech Synthesis System: http://www.cstr.ed.ac.uk/projects/festival/

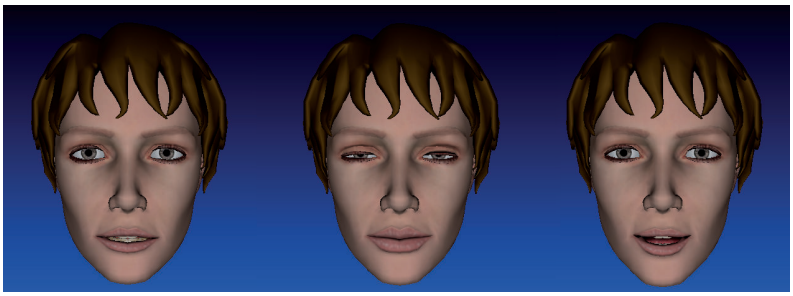

Figure 4.12: *Talking head, using AudioTTS and CoordinateMorpher nodes.*

The value of 'visemeDurationScale' determines how long the visemes are displayed during the animation. By default the value is 0.5, which means that half of the time the current viseme is displayed, a quarter of the time is used to interpolate from the previous viseme to the current viseme, and a quarter of the time is used to interpolate from the current viseme to the next viseme. Obviously the resulting curve is only a coarse approximation of real speech. Thence, a related and moreover language dependent topic is coarticulation, which denotes the variation that a speech sound undergoes under the influence of adjacent phones. A suitable mapping from phonemes to visemes in the case of German is described in [11], but an automated implementation is beyond the scope of this thesis.

Because within the X3D framework the viseme mapping is decoupled from the 'spoken' phonemes, a mechanism that is less error prone and easier to use has to be incorporated (see section 7.3.5.1), since precise synchronization and scheduling is needed for simulating virtual characters. The proposed animation controller component, which is described in section 7.3.4, is thus also capable of synchronizing the computer voice with the corresponding facial animations. Moreover, they are automatically combined with other morph targets (e.g. for displaying emotions), which are active at the same time.

4.4.2 Facial Animation

As stated in the previous paragraph, the *AudioTTS* node not only synthesizes speech, but also determines the resulting list of phonemes including the duration of every phoneme, and it also calculates the weights for the different visemes (as shown in Figure 4.12). Similarly, the weights can also be used to animate facial expressions in general such as the emotions shown in Figure 2.6 or weighted combinations of them. The weights can be used in two ways, either for animating the skin with the help of *HAnimDisplacer* nodes or by directly morphing the mesh, as is explained next [147, 149].

HAnimDisplacer nodes are usually used to control the shape of the face. Each *HAnimDisplacer* specifies a morph target that can be used to modify the displacement properties of the corresponding vertices. The scalar magnitude of the displacement is given by the 'weight' field and can be dynamically driven by an interpolator or a script. The combined displacements are added to their associated vertex before it is multiplied with the respective joint matrices for skinning. The mesh therefore can be morphed smoothly using the base mesh and a linear combination of all sets of displacement vectors.

Quite similar to the displacer node is the *CoordinateMorpher* node, another model-free

approach for doing animations that was first proposed in [4]. Assume you want to animate a face, and you have given n target states of your modeled face, a neutral one, and $n-1$ other ones, e.g. a smiling one, one with open and one with closed eyes, one with raised eyebrows, and the other ones for representing the phonemes. The morpher node now regards each of these states as a base vector of an n dimensional space spanning all possible combinations of mesh deformations. In order to get valid linear combinations the coefficients (weights) of all morph targets must sum up to 1.

For interpolating between different states additionally a *VectorInterpolator* node was introduced, because for each key time a vector of n key values is needed here. Unlike the *HAnimDisplacer* the morpher node is also suitable for animating a talking head or other objects not compliant to H-Anim. But as opposed to adding a weighted sum of displacement vectors onto the base mesh, this approach requires to separate the face mesh from the body mesh, which has several problems such as visible borders due to discontinuities in shading or even because the meshes are teared apart due to inaccuracies in the calculation. Additionally a *NormalMorpher* was defined here that linearly interpolates among the set of normals, whereas the *HAnimHumanoid*, besides the 'skinCoord' field, directly provides a 'skinNormal' field. These attributes are internally used to create the appropriate surface deformations and for updating the surface normals accordingly.

4.5 Hair Simulation

In this chapter we present a method for realistic rendering and simulation of human hair in real-time, which is suitable for the use in complex virtual reality applications [156], because the simulation is not only very robust but also easily parameterizable to ease the setup process. In this regard we also present our method for interactive hair styling [154]. Neighboring hairs are combined into wisps and animated with our modified cantilever-beam-based simulation system, which runs numerically stable and with real-time update rates [158]. The key point here is that despite external forces like wind and gravity the hair has to follow the head movements, which can result from dynamically combined or generated body animations. Further, the rendering algorithm adapts recent methods, utilizes modern graphics hardware features, and can even handle light colored hair by including anisotropic reflection and internal transmission to deliver high visual accuracy.

4.5.1 Introduction

There is a big trend towards the use of virtual characters for novel user interfaces. As mentioned, within several application areas visual realism of such characters plays an important role, especially when non-verbal communication gets important. To achieve this realism the visual appearance and movement of the character must be very close to reality, embracing skin, eyes, hair, clothing, gestures, locomotion etc. Modern film productions already show that it is possible to simulate virtual characters, which look completely real. But there are major drawbacks of the technologies used in this area: the algorithms are fairly slow, thus simulation and rendering of a single frame takes minutes and hours. Highly reactive user interfaces need real-time performance with an update

Figure 4.13: *Some frames taken from our real-time hair simulation (body approximated with four spheres for collision handling).*

rate of 24 fps or more. Thus animation and rendering must be very fast and should still leave a decent amount of processing power to the application.

Creation of realistic characters and their animation is a very time consuming task and can be carried out by specialists only. For a major break-through of realistic virtual characters for novel user interfaces it is inevitable that their creation gets simplified and animation as well as rendering performance significantly improved. Therefore, within this chapter we will focus on the realistic simulation and rendering of human hair. The simulation of human hair is still an open area of research in computer graphics, for that reason alone, that a person usually has more than 100,000 hairs. As of today the simulation and rendering of each single hair, together with all other aspects that make up a realistic virtual character, overstrains any common PC platform.

The problem is getting by far more complex, if hair simulation is only one aspect of the dynamic behavior of virtual characters, which also have to show natural movements, gestures, face expressions, and speech within their virtual environment. Additionally, besides the dynamic simulation of hair the realistic rendering of skin, eyes, and certainly hairs is required (in Figure 4.13 some frames from our simulation are shown). Even with recent CPUs and programmable GPUs many simplifications have to be made to be able to perform all these tasks together in real-time in a robust and easy to use manner. Hence, the demands of information rich dynamic virtual environments in general, like moving cameras, light sources, and avatars, raise a lot of new problems with respect to both rendering and simulation: e.g. collision detection has to deal with moving body parts etc.

Thus for the use in real-time applications the large number of hairs and the many types of interdependencies like hair-body, hair-hair and hair-air interaction call for simplifying solutions. Many simulation methods are based on mass-spring-damper systems, in which hair strands are modeled as line segments connecting some mass points, and the animation is done by iteratively computing the forces acting on them. The arising differential equations can be solved with explicit ODE solvers, which do not demand much computational power, but tend to blow off when forces become too strong. Implicit methods on the other hand like the backward Euler method are stable, but they are based on the computational expensive evaluation of big matrices, and constraints resulting from collisions make things even worse. Besides taking care of numerical stability there is also the

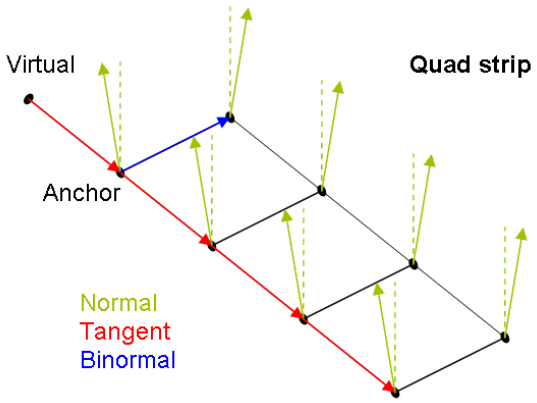

Figure 4.14: *Quad strip based hair model.*

problem of an adequate parameterization, otherwise the hair movement might resemble rubber bands or look like in an under water scenario.

Insofar we propose a wisp hair model based on quad strips (Figure 4.14), which is animated by a simplified cantilever beam simulation. Rendering is done with GLSL shaders [266] and includes amongst other things anisotropic reflection and a second specular highlight, which is characteristic for light colored hair. Because we are mainly focusing on simulation and rendering, the modeling aspect will only be covered shortly in the next section. In section 4.5.3 we will describe our simulation system, and section 4.5.4 deals with rendering aspects like geometry sorting and lighting models.

4.5.2 Modeling and Styling

In order to reduce the geometric complexity and to avoid rendering problems we model hair wisps as relatively small quad strips instead of generating every single strand (see Figure 4.14) [158, 154, 156]. Although polylines seem to be the natural representation of hairs at a first glance, they not only increase the load of the vertex processor, because a lot of lines are needed for a volumetric appearance, but also have to deal with aliasing artifacts due to the long thin nature of a hair which usually when projected is thinner than a pixel's size. Because of our fast simulation algorithm (see next section) we can model each hair strip from many small segments. Therefore we don't need smoother parametric representations like NURBS patches [185] and can take advantage of directly using the OpenGL support for rendering polygonal primitives.

The quad strips are appropriately layered on top of the scalp mesh for providing an impression of volumetric qualities. They can consist of a variable number of segments along the direction of hair growth, which is specified by the position of the hair root and an initial hair tangent vector. The hair distribution is defined by selecting some surface polygons on the scalp mesh. After having specified all other necessary parameters like width, height and number of segments as well as hair density the hair style is generated conferring to these parameters.

To allow users modeling different hair styles, we developed a hair creation plug-in for the

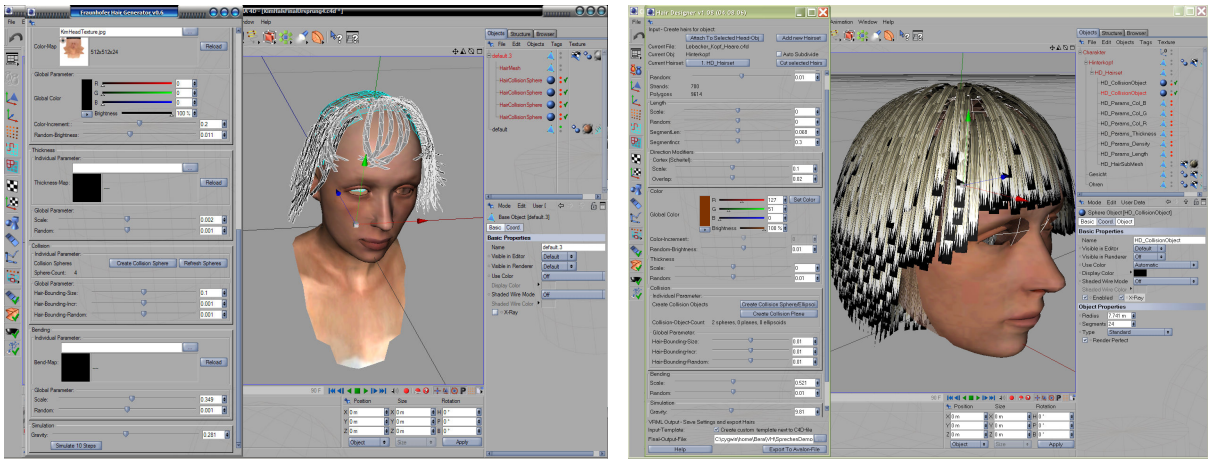

Figure 4.15: *Older texture-based version (left) and new geometry-based version of the HairDesigner tool (realized as Cinema4D plug-in) for creating a hair style.*

3D modeling software Cinema 4D [216] (see Figure 4.15) for simplifying the process of creating different hair styles. It allows loading a human head and then placing interactively hair strands on top of the head. The base color, length and direction of each hair strand can be adapted individually. In addition, selected hairs can be cut. Furthermore, to ease the modeling process, we have implemented a texture-based tool (where texture maps are used for a first parametrization of hair density or color as proposed in [126]) and a geometry-based modeling tool (Figure 4.15, right), in order to be able to compare and evaluate both methods. In both cases, standard GUI elements with direct visual feedback such as sliders are used for the fine adjustment of simulation parameters.

In our user studies the latter method yielded to better and faster modeling results, because the body paint tool of Cinema 4D, which we used as basis for the texture-based modeling tool, was neither very intuitive to use nor did it correctly work with unsuitable texture coordinates, which unfortunately can be often found at the back of the head. Since the hair styling highly depends on the dynamic simulation, the whole simulation system was embedded in the plug-in, thus the user can immediately experience the results of his modeling and how it will look like in the virtual reality environment. Thereby users are able to create realistic hair styles within a couple of minutes. The final result of the modeling can then be exported as an X3D file.

4.5.3 Dynamic Simulation

4.5.3.1 Structure

Our hair simulation is derived from the cantilever beam method [131], which originally was intended for hair modeling only but not for its dynamics simulation. Compared to mass spring approaches, it provides a numerically simpler and – at least for smooth hair styles – visually more convincing way to simulate hair.

As already mentioned, mass-spring-systems are used for hair dynamics frequently. Mathematically these can be seen as damped oscillators which can be calculated with the help

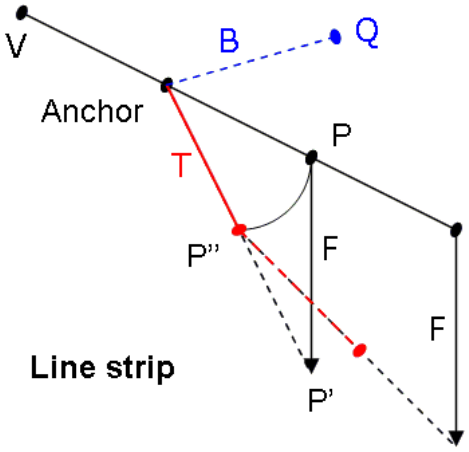

Figure 4.16: *Cantilever beam like simulation structure consisting of several segments.*

of differential equations by equating Newton's second law of motion ($F = m \cdot a = m \cdot \ddot{s}$) and Hook's law ($F = k \cdot s$), which relates the force F exerted by a spring with spring constant k and rest length l to its deflection $s = l' - l$.

Explicit numerical methods for solving these differential equations do not necessarily converge if forces are too strong and the size of the time step Δt lies above a certain threshold [18]. Whereas Anjyo et al. also used differential equations for computing the iteration steps during animation, our cantilever beam method works without them, which results in much higher simulation speed making it applicable for real time dynamics.

The most important difference of kinematic models like the cantilever beam model compared to an ordinary mass spring system is that the initial distance l between connected vertices can be fully conserved. Because neighboring elements don't interact by means of spring forces, oscillations cannot occur. Thus a kinematic simulation system keeps stable even with much bigger time steps.

Our modified cantilever beam algorithm internally works on a kinematic multibody chain, as illustrated in Figure 4.16. The nodes of the multibody chain are defined by the vertices of the original geometry and can be seen as joints connecting the edges between them. Two different types are distinguished, anchors and free moving vertices. Anchors, resembling the hair roots, are connected to the scalp and serve as the attachment point of the chain, whereas all the other vertices in the chain are free moving.

Free moving vertices are moved only due to external forces like gravity, the bending forces caused by their connected neighbor vertices and by applying the length conservation constraint. An external force F, e.g. gravity, which is acting on a chain link, results in a bending moment M, that causes a deflection of the actual segment along the direction of F. The calculation of the effect of this force is simplified by means of a heuristic approach: instead of calculating the torques, all forces, which are acting on the succeeding segments of a cantilever beam, are expressed by adding some offset vectors. Then length conservation is achieved by simply scaling the resulting vector back to the rest length l.

This method yields another advantage. During motion of the head, i.e. a coordinate

Figure 4.17: *Some screenshots showing a character that runs his fingers through his hair, which is approximated with a collision sphere (cp. section 4.5.3.2).*

transformation $\mathbf{T}$ of the head's frame of reference, it is not necessary to derive forces from $\mathbf{T}$ and feed them into the simulation to move the hair roots. It is sufficient to transform the hair root A directly accordingly to the new position ($A' = \mathbf{T} \cdot A$), because the transformation is then propagated through the chain intrinsicly keeping the hair length constant. While gaining a noticeable improvement in performance this way furthermore there are no stretching artifacts even when abrupt motions happen. Additionally, to represent the inner tension of hair, the angle between the bent hair segment and the direction of the previous hair segment is scaled down by a bend factor smaller than one. This leads to very realistic looking hair motions.

For stabilizing the orientation of a hair wisp and aligning the beginning of the chain during movement according to the direction of hair growth a virtual vertex fixed to the scalp is introduced as predecessor of the anchor point (as shown in Figure 4.14). Together with the anchor it defines the growth direction during simulation. This way the hair strip's binormal B can be aligned horizontally to the head.

4.5.3.2 Collision Detection and Response

As has been stated earlier, besides a convincing simulation method a natural behavior in case of collisions is also required. Concerning hair, collision detection can be divided up into two types of interdependencies: hair-body and hair-hair interaction. Because of the large amount of hair, the trade-off between quality and speed of the collision detection has to be taken into account.

Collisions with head or body are a hard constraint and must be treated explicitly. Tests have shown that because of the high self-occlusion of hair, users usually take no notice of a relatively low accuracy in collision detection between hair and head. Thus for approximation of the head we use parametric collision objects like spheres, ellipsoids and planes, for which intersection tests can be handled quite efficiently. If a free moving vertex moves into a collision object, a penalty force is determined that projects the vertex back onto the surface. In addition, we have also successfully used such collision objects for simulating certain hair styles like the brushed-back hair shown in Figure 4.24 (right).

Hair-hair collision can't be handled easily in real-time. Thus the inter-penetration of hair wisps is avoided with a trick. On the one hand, hair strips are arranged on top of the scalp in different layers, each within a different distance to the head. For keeping this up during dynamics, each vertex P, depending on its position, is assigned a virtual collision sphere

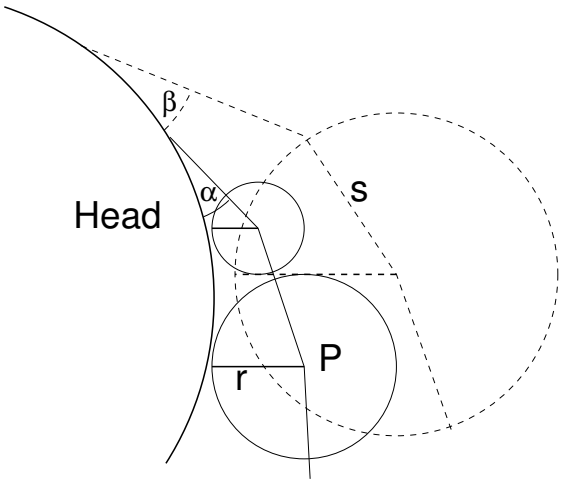

Figure 4.18: *Collision handling with collision spheres per strand vertex.*

with a different radius r_P, in order to parameterize the distance to the head individually. On the other hand the problem is alleviated by using a slightly different bending factor for every chain, based on the position of its respective anchor.

This layered collision avoidance structure is illustrated in Figure 4.18. As can be seen, hairs from lower levels are not bent as much as those from higher levels. Besides this the vertices which are located nearer to the hair tip or which belong to hair wisps layered on top of the head are assigned bigger collision spheres than those from the bottom hair. This also has the nice side effect that implicitly collisions of hair segments s with the head are handled too, albeit with lower accuracy, although the algorithm explicitly only regards the vertices P.

4.5.3.3 Algorithm

As aforementioned the hair simulation is calculated on a skeleton, defined by one longitudinal edge of the quad strip. In order to re-transform the chain structure into a polygonal structure, every joint P_i has an associated point Q_i, which belongs to the second longitudinal edge. The vertices Q_i are calculated by adding the binormal vector B (the blue one in Figure 4.16) scaled by the initial strip width to the vertex P_i.

Normal, tangent and binormal are not only needed during rendering (see next section) but also for the simulation [158, 156], which is done iteratively and consists of the following steps for the current time interval Δt:

1. For a simulation speed independent from the frame rate, scale the force offset vectors with an averaged time interval Δt_{avg}.
2. Transform all anchor points A, virtual points V and collision objects k to the new world coordinates.
3. $\forall$ anchors A calculate associated point Q, T_{avg} (difference vector between the last joint of a chain and A), tangent T, binormal B, and normal N:

$$T = A - V;\ B = T \times T_{avg};\ N = T \times B$$

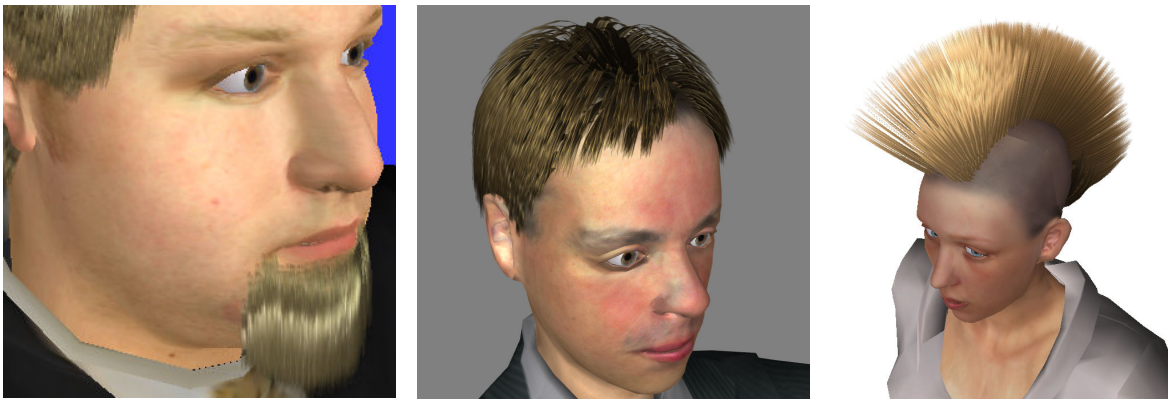

Figure 4.19: *Left: light hair with beard. Middle: short brown hair. Right: freaky hair style.*

4. $\forall$ joints P_i of every anchor A:
   a) add offset vectors derived from forces
   b) $\forall$ collision objects k with radius r_k: if P'_i is inside k then project P'_i back onto the surface at distance $r = r_k + r_{p_i}$
   c) add bending offset vector to P'_i
   d) calculate vectors T_i and N_i:

$$T_i = P'_i - P'_{i-1};\ N_i = T_i \times B$$

   e) keep initial segment length l_i:

$$P''_i = P'_i + T_i(\frac{l_i}{|T_i|} - 1)$$

Because each quad strip is handled separately, the algorithm can be parallelized easily e.g. with OpenMP,[4] which now is part of every recent C++ compiler. This can be realized by adding one single line of code directly before the outer `for` loop that is responsible for updating the anchors and their associated chain:

```
#pragma omp parallel for
```

Thereby, on a dual core CPU a doubling of performance is achieved. Also, by having a separate application and render thread, some speed improvements can be obtained especially for older Shader Model 2.0 hardware, which was rather fill-rate limited.

4.5.4 Rendering

4.5.4.1 Sorting

In order to avoid aliasing and to overcome the rectangular structure during rendering, alpha blending, with disabled depth buffer write, is used in combination with usual RGBA

[4] http://openmp.org/wp/

textures, which are mapped onto the hair patches. In this way, an impression of thin, semi-transparent hair is created. Alpha blending only works if the hair geometry is rendered last and sorted back to front along the viewing vector, otherwise the edges of the underlying geometry, like the head, would be visible. Because neither the viewpoint nor the virtual character can be assumed to be immobile, sorting has to be done dynamically.

It is obvious that a naïve per quad sorting must fail, because the hair patches are very densely neighbored. Besides this, the algorithm cannot always guarantee that there is no self pervasion. Insofar no unique sorting sequence can be given, which takes the connection of the segments of a hair patch into consideration. A simple solution to this problem is to sort the primitives at the next level of hierarchy.

Back to front sorting of the quad strips solves the latter problem but arises another artifact, caused by the relatively large extension of the surface compared to one single reference point chosen for sorting purposes. Then the roots of wisps, which are lying nearer to the camera, seem to be layered on top of hair wisps being further away, although in reality the hair position is view-independent. This can be alleviated by sorting the uppermost strips along the head's up-vector in a second step. To our experience an empirically determined factor of about fifteen percent of all non-occluded hair strips produces quite satisfactory results.

Alleviating Sorting Artifacts

As already mentioned, to create an impression of thin, semi-transparent hair, textures with hair-like transparent patterns are mapped onto the hair patches. To provide an impression of hair volume, alpha blending is used, which requires correct back-to-front sorting of the hair wisps. However, for simulated quad strips no unique sorting order can be determined and therefore severe sorting artifacts may result. As proposed in [276], most sorting artifacts can be alleviated by a multi-pass approach.

Although this method is only suitable for pre-sorted hair without animation, in combination with our previously explained method for rendering human hair it leads to pleasing results. After rendering all back-facing polygons with depth writes disabled and depth test set to "less", the front-facing polygons are rendered. This is accomplished by means of our *MultiPassAppearance* node and the additional use of special render mode nodes like the *FaceMode* node and the *DepthMode* node for fine grained render state control (the proposed node extensions are described in section 6.3.4).

4.5.4.2 Lighting Model and Shading

Reflection Properties of Human Hair

The long thin nature of hairs, which usually are aligned in one predominant direction and microscopically consist of step-like seceding segments (see Figure 3.10, right, p. 85), can be regarded as the micro-structure of a hair style. It therefore contributes to the hair's anisotropic reflection properties, because the normal distribution along the hair fibers is different from the distribution across them. The first impressive results for hair rendering already have been achieved by Kajiya and Kay [165], who also suggested a very common

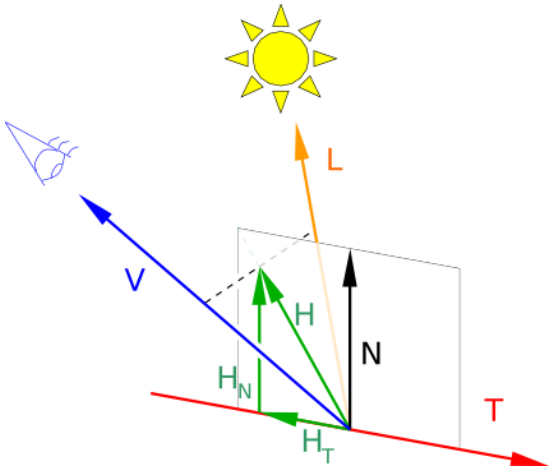

Figure 4.20: *Tangent based lighting model (the tangent is given by the hair direction).*

phenomenologically motivated local lighting model, which still constitutes the basis for many modern rendering approaches concerning hairs.

The demand for photorealism however calls for a shift from phenomenologically-based lighting models to physically correct lighting simulations. Despite the great advances in the field of graphics hardware, there still remains a trade-off between visual quality and rendering speed. Because anisotropic reflection cannot be evaluated with standard fixed-function methods, nowadays rendering is directly done on the GPU using hardware shaders, with which nearly photo-realistic real-time effects can be reached.

Hair fibers can be regarded as cylinders which are nearly infinitesimal small in diameter. Unlike it's the case with surfaces for a given point P on the hair fiber there exists an infinite number of normals lying in a plane orthogonal to the fiber's tangent T. A normal N suitable for lighting calculations is the normal which is coplanar with the half vector $H = \frac{L+V}{|L+V|}$ and the tangent T (see Figure 4.20).

This normal vector does not need to be computed explicitly for calculating the intensity of the specular highlight, described by $(H \cdot N)^s$ in the Blinn-Phong illumination model [29], when making use of an orthogonal decomposition of the half vector $H = H_N + H_T$. Then the specular term can be calculated solely in terms of H and T (both being of unit length) as follows [356]:

$$H \cdot N = |H_N| = \sqrt{1 - |H_T|^2} = \sqrt{1 - (H \cdot T)^2}$$

Specular Highlights

The rendering of long, light colored hair is by far much more complex than that of short, dark hair. Thus, due to the translucency characteristics of hair fibers the additional consideration of transmission, dispersion and self shadowing is required. As described in Marschner et al. [213], in case of direct lighting, there are two different specular highlights. The first highlight results from direct reflection R at the surface of a hair fiber. Caused by the thin tilted squamous structure it steps somewhat shifted up along the tangent towards the hair root.

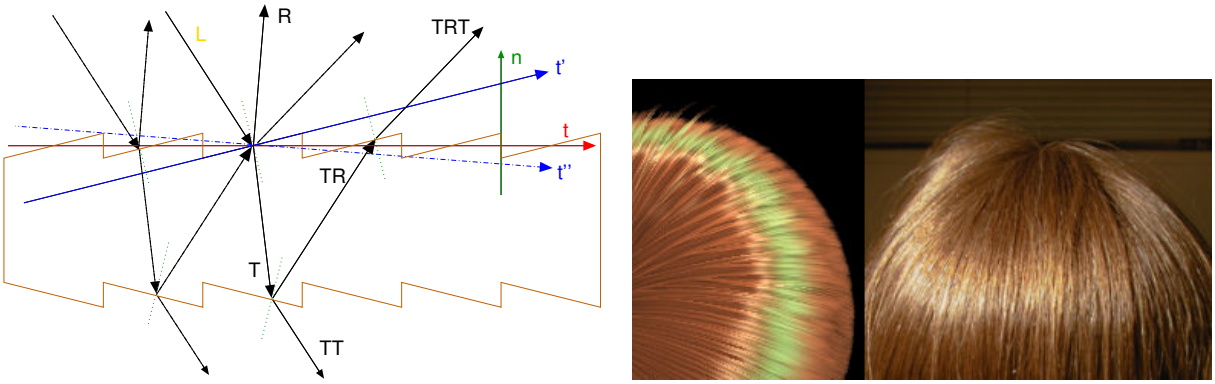

Figure 4.21: *Left: transmission T and reflection R in a hair (after [213]). Right: virtual hair (second highlight for visualization in green) compared to photo of real hair.*

The second highlight TRT, which does not arise with black hair, results from internal reflection (see Figure 4.21, left). The incident light passes through the interior of the fiber and is reflected at the opposite side of the cylindric shape. Because of refraction the light's direction changes when passing through two media with different densities, so the secondary peak appears enervated and shifted towards the hair tip. The highlight is more like a glint and because of its way through the medium it gets colored by the pigments of the hair. Likewise only with lighter hair the transmission-transmission term TT produces backlighting effects.

A physically correct treatment exceeds the capacities of a real time application. Nevertheless, in order to calculate the different peaks described above (second highlight shown exaggerated in green in Figure 4.21 on the right), after Scheuermann [276], two tangents T' and T'' are needed, which are shifted in opposite directions. This can be achieved by adding a scaled normal onto the original tangent T (diagrammed in Figure 4.21, left), given by the hair's direction, which is updated during the simulation anyway. The observed dispersion of light, caused by scattering in the interior of the medium, is simulated by a noise function which for efficiency reasons is stored in a texture and can easily be accessed by a simple texture look-up in the fragment shader.

To achieve best results, the rendering equation is calculated per fragment on the GPU. For implementation the *ComposedShader* node of the X3D Shaders component was used. The vertex shader computes all necessary vectors and passes them to the fragment unit. In Listing 4.1, variables with the prefix v indicate (GLSL-specific) varying parameters, which are passed over from the vertex to the fragment processor. The pixel shader calculates the diffuse and specular term for the lighting model and another term for describing the ambient light that has been scattered around for several times.

Ambient and Diffuse Lighting

Scattered light, which leaves the backfacing side of a hair fiber (see TT in Figure 4.21, left), is simulated by a scattering function on the GPU by extending the diffuse Lambert term $N \cdot L$. The trick with the so called wrap or rim lighting is to "turn" the light around the object. Therefore a small term $w \in [0; 1]$, which can be constant or better based on the angle between the viewing and normal vector, is added to the diffuse term (see line

```
1  vec4 hairTexCol = texture2D(hairTex, gl_TexCoord[0].st);
2  if (hairTexCol.a <= 0.1)
3    discard;
4  else {
5    float shiftTex = texture2D(noise, gl_TexCoord[0].st);
6    float wrapDiffuse = max(0, (dot(vL, vN) + scattering) / (1 + scattering));
7    vec3 diffuse = wrapDiffuse * diffuseColor;
8    vec3 ambient = ambientColor * vC;
9    vec3 specular1 = specularColor1 * calcHighlight(
10           vT, vN, vH, shininess1, shiftValue1 + shiftTex);
11   vec3 specular2 = specularColor2 * shiftTex *
12           calcHighlight(vT, vN, vH, shininess2, shiftValue2 + shiftTex);
13   vec3 color = (specular1 + specular2 + diffuse + ambient) * hairTexCol.rgb;
14   gl_FragColor = vec4(color, hairTexCol.a);
15 }
```

Listing 4.1: *GLSL fragment shader code snippet for hair rendering.*

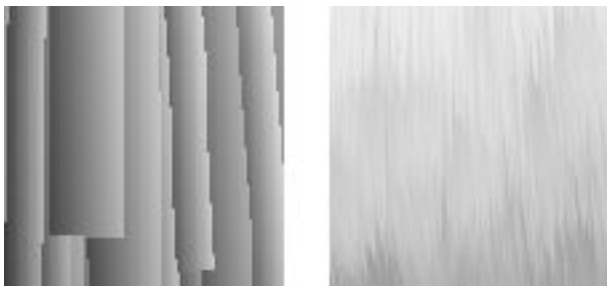

Figure 4.22: *Result of normal bending (left) and simulated ambient occlusion (right).*

6 of Listing 4.1). This causes a certain light intensity also on the backfacing side (cf. Fernando et al. [81, p. 264]). To improve the impression of a hair volume, the normals of each patch are bended outwards into the direction of the binormals (see Figure 4.14, and Figure 4.22, left). Thus the light intensity is sloping towards the edges, which leads to the impression of a cylindrical and thereby more volumetric shape.

A technique for approximating complex light distributions is called 'ambient occlusion' (AO) [81]. The basic idea is to calculate for each point to be lit the fraction of incident direct light that is not occluded by any other geometry. This self-shadowing information will be used to scale the lighting term. The original method consists of several rendering passes, nevertheless it can be simplified with some object knowledge, thus hair from lower layers receive less energy than those from top layers. This occlusion term is calculated per vertex depending on its position and the height of its associated anchor point (see line 7 of Listing 4.1, and right side of Figure 4.22).

With help of the fragment processor's *discard* command (line 3 of Listing 4.1), fragments which satisfy the condition $\alpha < \varepsilon$, with a certain $\varepsilon > 0$, are thrown away. Defining areas in the alpha map which shall be ignored in further lighting calculations not only saves computation time but also has another advantage. With simple alpha blending and no special shadow pass materials, shadows which are cast from the hair patches onto the character's head would be shaped like a whole patch instead of single hairs, because transparency is disregarded in depth mapping. By discarding certain pixels, good shadowing results are achieved even with simple depth maps. Note that for the shadow pass no sorting must be done to avoid popping artifacts.

4.5.5 Interfaces and Usage

The simulation and rendering components were implemented in C++ as native scene graph nodes in our VR/AR system Instant Reality [135], which supports an extension of X3D as the application description language. This is done by defining a simulation system scene-graph node and another scene-graph node for rendering a set of sorted primitives (shown below), whose field values are updated by the simulation system node via the X3D routing mechanism. Therefore the simulation system as well as the hair shaders are easily to create, use and parameterize even during runtime.

The rendering component, a standard X3D *Shape* node consists of the hair appearance and the *SortedPrimitiveSet* node. It holds all geometric properties like positions, indices and tangents and is responsible for the CPU based part of the sorting algorithm. The latter can be parameterized by the 'drawOrder' field, the 'upTheshold' field for defining the threshold for the second sorting step as explained in section 4.5.4.1, and the 'lowerBound' field for determining the percentage of quad strips which for further speed-ups can be omitted after sorting during rendering due to occlusion.

Because of its generic design our proposed *SortedPrimitiveSet* node likewise is useful for similar usages like rendering grass. It is updated via the X3D routing mechanism by the simulation component. This way the shaders only belong to the appearance nodes and are therefore interchangeable and easily to parameterize.

```
SortedPrimitiveSet : X3DComposedGeometryNode {
  SFString [in,out] mode        "QuadSet"
  SFString [in,out] drawOrder   "BackToFront"
  SFNode  [in,out]  refPoint    NULL
  SFNode  [in,out]  coord       NULL
  SFNode  [in,out]  color       NULL
  SFNode  [in,out]  normal      NULL
  SFNode  [in,out]  texCoord    NULL
  SFNode  [in,out]  tangent     NULL
  MFInt32 [in,out]  index       []
  SFFloat [in,out]  lowerBound  0.25
  SFVec4f [in,out]  upThreshold 0 1 0 0.85
}
```

Obviously a similar topic is cloth rendering and simulation, where for simulation generally mass-spring-systems are used. In [90] one of the biggest problems in that area, the collision detection, is discussed and at least for rigid objects solved by using distance fields. For rendering fabrics, many approaches were proposed, e.g. the Oren-Nayar reflectance model [2, p. 262] as an extension of the Lambertian model for simulating diffuse lighting for rough surfaces, or approximated BTFs as presented by Kautz [172].

In [156] we have presented an example with additional per vertex displacement based on the evaluation of a height map in the vertex shader for improved realism even in close-up views of corduroy fabric.[5] Although displacement mapping leads to very convincing

[5] Textures were taken from http://btf.cs.uni-bonn.de/index.html

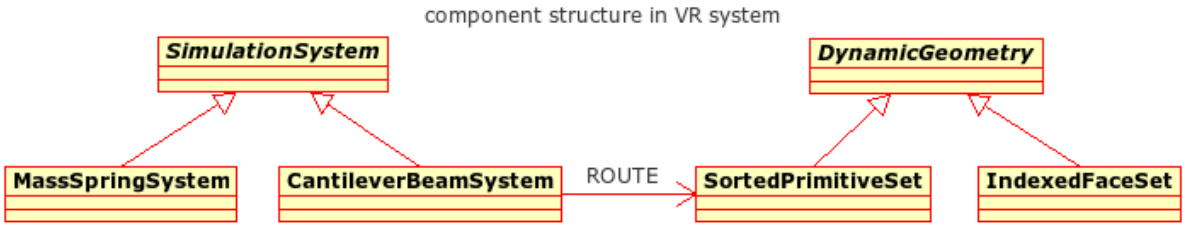

Figure 4.23: *UML diagram showing the structure of the proposed X3D components.*

results for many materials it is not suitable for cloth rendering in complex environments because the geometry needs to be highly tessellated. But in combination with progressive meshes and the Shader Model 4.0 graphics hardware, which additionally supports geometry shaders that are able to create and delete triangles on the fly, this approach can be extended and made feasible for real-time applications.

```
SimulationSystem : X3DNode {
  SFBool  [in,out]  enabled          TRUE
  SFTime  [in,out]  time             0
  SFFloat [in,out]  speed            1.0
  SFInt32 [in,out]  minStepsPerFrame -1
  SFInt32 [in,out]  maxStepsPerFrame -1
  SFTime  [in,out]  maxStepTime      -1
  SFBool  [in,out]  localCoordSystem TRUE
  MFInt32 [in,out]  index            []
  MFInt32 [in,out]  anchorIndex      []
  MFVec3f [in,out]  coord            []
  MFVec3f [in,out]  normal           []
  SFVec3f [in,out]  gravity          0 -9.81 0
  SFVec3f [in,out]  externalForce    0 0 0
  SFFloat [in,out]  massRadius       1.0
  SFFloat [in,out]  staticFriction   0.5
  SFFloat [in,out]  slidingFriction  0.5
  SFFloat [in,out]  airFriction      1.0
  MFString [in,out] collisionDomainType  []
  MFFloat [in,out]  collisionDomainParam []
}
```

In this spirit we also propose some simulation system nodes. The *MassSpringSystem* node is suitable for applications like cloth simulation or e.g. gelatinous materials but also for simulating the dynamics of hair. As can be seen on the left side of Figure 4.23 this node inherits from our abstract *SimulationSystem* base node type and is thereby strictly separated from the geometry and appearance components for maximal flexibility. Because for different applications different integrators may be required (section 4.5.6.2), the user can set these with help of the 'integrationType' field (e.g. explicit Euler scheme). The node also provides fields for parameterizing the spring constants, damping factors etc. In addition, the node's 'set_trackPoint' eventIn slot can be used for applying "user" forces to certain mass points, for example when dragging a piece of rag.

Figure 4.24: *Light straight hair. From left to right: blond hair viewed from behind, gray hair with fringe (both circa 37,000 vertices), brushed-back hair (around 23,000 vertices).*

The *CantileverBeamSystem*, whose node interface is shown below, also inherits from the *SimulationSystem*. With the 'collisionDomainType' and 'collisionDomainParam' fields, the collision objects can be specified. Both attributes are defined aligned with the corresponding domain parameters of the "Particle System API".[6] Because full triangle tests would be too expensive, the 'massRadius' field can be used for setting the distance between connected vertices as was described in section 4.5.3.2. Furthermore, the fields 'minBend' and 'maxBend' define the bending parameters as depicted in Figure 4.18.

```
CantileverBeamSystem : SimulationSystem {
  MFVec3f [in,out]  tangent  []
  MFColor [in,out]  color    []
  MFVec2f [in,out]  texCoord []
  MFVec3f [in,out]  refPoint []
  SFFloat [in,out]  minBend  0.05
  SFFloat [in,out]  maxBend  0.10
}
```

4.5.6 Results and Discussion

In contrast to the often used mass-spring-systems our heuristically motivated hair simulation system is computationally less intensive because of the special chainlike structure of the hair wisps. Especially properties like pliancy can be easily included without any additional expense. Moreover the system is nearly non-oscillating and insofar it is most unlikely that it will blow off. Therefore the number of iterations per time-step can be reduced a lot in favor of a much higher frame-rate. Concerning the lighting simulation, we are likewise still working on a phenomenological level by adopting approaches from physically based rendering to simpler GPU-based methods. However, the result is very close to photorealism and is applicable in information rich and highly dynamic virtual environments. As can be seen in Figures 4.24 and 4.27 (which also shows some real photographs of moving long, blond hair for comparison), a realistic appearance with interactive frame rates is achieved even for light colored hair.

[6] Particle System API by David K. McAllister, http://www.particlesystems.org/

	Single Core PC + SM 2.0 GPU				Dual Core + SM 4.0 GPU		
# Vertices	5096	18690	37498	52192	23182	37498	52192
# Anchors	364	623	2316	2388	1511	2316	2388
Simulation	248 fps	80 fps	37 fps	18 fps	600 fps	294 fps	204 fps
Rendering	64 fps	49 fps	19 fps	16 fps	312 fps	172 fps	101 fps
Combined	52 fps	35 fps	15 fps	11 fps	125 fps	83 fps	55 fps

Table 4.1: *Benchmarks for different hair styles (on the left with Single Core PC and SM 2.0 GPU and on the right side with Dual Core Laptop and SM 4.0 GPU).*

4.5.6.1 Benchmarks

Benchmarks are shown in Table 4.1 for four different numbers of vertices resp. anchor points. The values to the left have been taken using a Linux PC with Pentium IV, 2.8 GHz with Hyperthreading, 1 GB RAM with a nVidia GeForce FX 5900 Ultra graphics card (about 170 000 pixels are covered by hair), and the values to the right were obtained using a Linux Laptop with Intel Core 2 Duo CPU, 2.53 GHz, 4 GB RAM and a nVidia GeForce 9600M GS GPU. The first row shows animation performance with standard OpenGL lighting, the second shows performance with GLSL shaders but without simulation, and the last row shows frame rates achieved with both simulation and shaders activated.

Figure 4.25 visualizes the runtime comparison of different hairstyles (including a beard) as shown in Figure 4.26. Again the benchmarks were obtained on two different systems, both running Linux. System **A** (openSUSE 11.0) is an old Intel Pentium IV desktop PC, with 2.4 GHz, 1.5 GB RAM, and with a nVidia Quadro FX 1100, which like the GeForce FX 5900 Ultra is also only Shader Model 2.0 capable. System **B** (Ubuntu 8.10) has a newer Intel Core 2 Duo T9400 CPU, 2.5 GHz, 4 GB RAM, and a nVidia GeForce 9600M GS that provides Shader Model 4.0 but nevertheless is only a slimmed-down laptop GPU.

4.5.6.2 Comparison with other Approaches

For comparison additionally the simulation times for a different hair style (the leftmost in Figure 4.26) were measured that was simulated with the help of a simple mass-spring-system using the explicit Euler scheme. As opposed to the one-dimensional simulation skeleton explained in section 4.5.3, here we also allow for a two-dimensional structure (cp. Figure 3.3, left) for being able to deal with other base hair types as e.g. used in [191, 276].

Hence, this hair model consisted of 23 *IndexedFaceSet* nodes (altogether 1311 vertices, 34 faces per strip and an extent of two faces along the strips' width). In addition, each geometry node is transformed differently to allow per strand sorting with the standard X3D approach. By using only one collision ellipsoid and a suitably small time step with only standard OpenGL lighting enabled, the pure simulation took around 28 ms on the old system **A** and 14 ms on the more recent system **B**. By increasing the strand count to 228 transformed *IndexedFaceSet* nodes and 5096 vertices (in Figure 4.26 the second hair style), for one frame system **A** already needed 66 ms. Again the simulation acted on all vertices, but this hair style was modeled according to Figure 4.14.

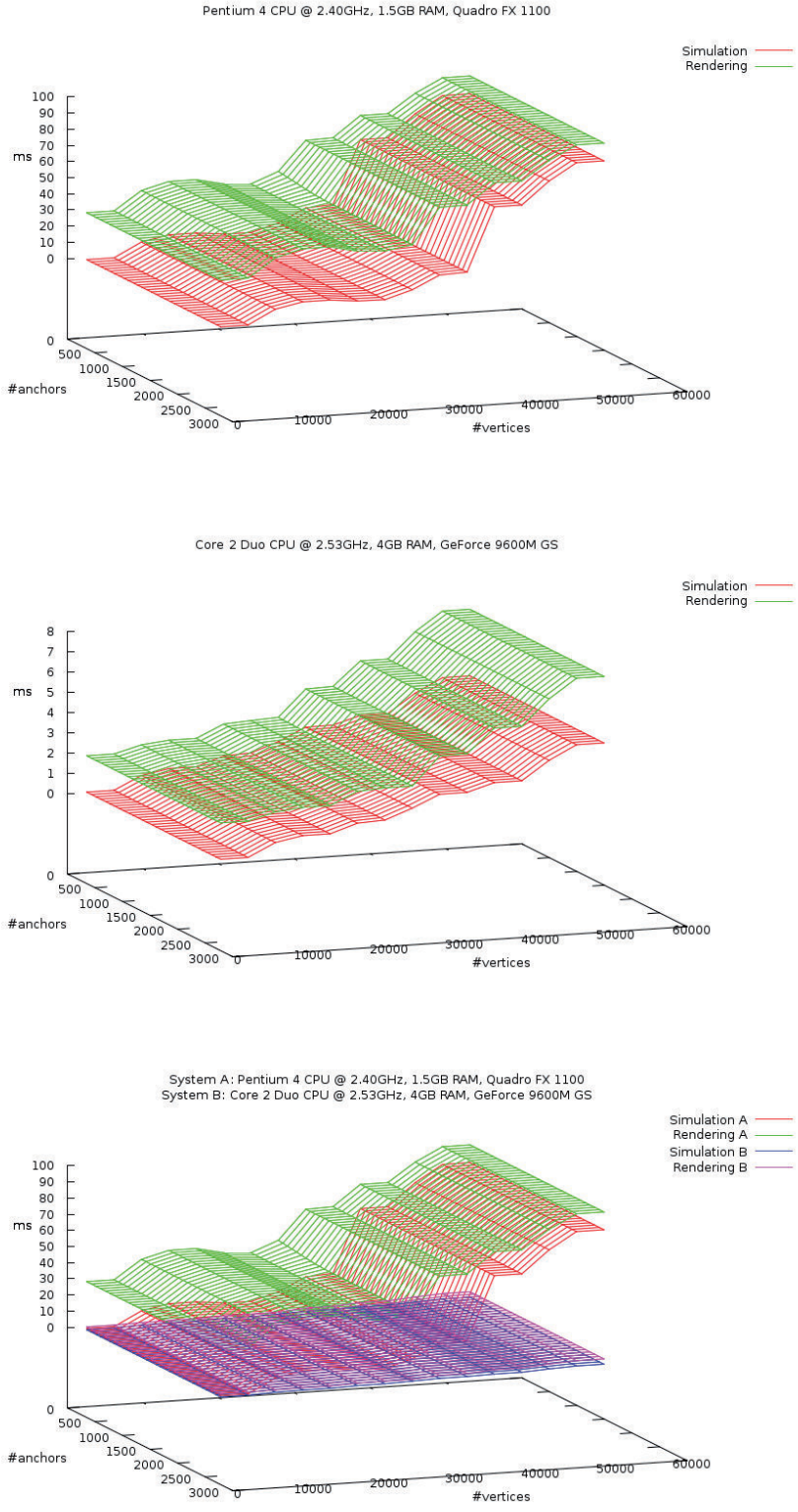

Figure 4.25: *Runtime comparison of hair simulation and rendering for two different systems: System A: Pentium 4 CPU @ 2.40GHz, 1.5 GB RAM, nVidia Quadro FX 1100. System B: Core 2 Duo CPU @ 2.53GHz, 4 GB RAM, nVidia GeForce 9600M GS.*

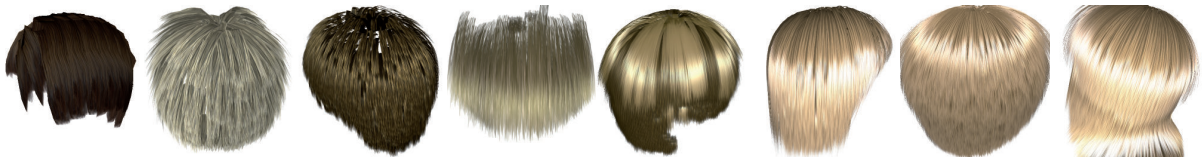

Figure 4.26: *The hair styles used for the benchmark shown in Figure 4.25 (the leftmost hair style was simulated with a mass spring system for comparison).*

Though it was still interactive, the main drawbacks of this method were its numerical instability, especially when high enough frame rates can't be guaranteed, as well as the method's rather difficult and unintuitive parameterization options. Assigning values to the different spring constants ("structural", "bend", "shear", and user defined ones in case interaction with hairs shall be possible – cp. Figure 3.3, left, p. 69) as well as other parameters exposed by the aforementioned *MassSpringSystem* node, such as damping factors, friction or restitution, is mainly a process of trial and error and hence neither evident nor intuitive to the user. Once suitable values have been found, the simulation results appear plausible for very small head movements, but for stronger movements they are similar to floor cloths under water, which motivates the reasons why for the NVidia Nalu Demo [224] an underwater scenario was chosen.

Adaptive Level-of-Detail Simulation

To prove that these issues not simply resulted from this naïve and unoptimized approach, for comparison we have also implemented an adaptive level-of-detail approach following Ward et al. [332, 333]. Our simulation is likewise based on a mass-spring-system model including two more advanced integration schemes, namely Verlet [346] and, to achieve greater stability while taking larger time steps, implicit Euler [18, 346]. Basically, this adaptive hierarchical simulation algorithm emphasizes the dynamic transition from clustered hair wisps to single hair strands in consideration of external factors like viewing distance or speed of movement. Depending on these factors, the hairs are represented as strips, clusters or strands as shown to the right in Figure 3.11 on p. 86 in section 3.4.

To represent the simulation skeleton as well as its geometrical representation we have designed a *Skeleton* node whose coordinates specify the initial hair strip of one cluster including its structure as necessary for dynamics simulation. Each such base hair has the derived classes Strip, Cluster, and Strand. For rendering, the quadtrees that are precalculated during initialization are represented by an OpenSG sub-graph. An important design decision is the choice of the method for doing numerical integration. A method like the implicit Euler integration is computationally more costly then simpler techniques, since here additionally two Jacobian matrices that represent the spring connections have to be created and multiplied, and a linear equation system has to be solved.

Hence, internally a base hair instance aggregates a mass-spring-system object, which likewise references an integrator instance with three derived classes for Verlet integration and for explicit and implicit Euler integration. During runtime the refinement depends on the aforementioned subdivision criteria, whereas the combined velocity of a hair node results from the last integration step. Thus, in case Verlet integration was chosen, the modified Velocity-Verlet should be used. To avoid instabilities and simulation errors, the

mass distribution in the tree has to be kept constant, which is achieved by quartering the masses when splitting. Moreover, popping artifacts can occur during level switches.

For comparison we have evaluated several factors, using a one-dimensional, line-strip-like simulation skeleton for numerical integration. First, we compared the three integration methods concerning runtime performance and stability. Whereas explicit Euler and Verlet integration barely differ in execution speed, implicit Euler takes around three times longer. Most interestingly, explicit Euler allows slightly bigger step sizes than Verlet until instabilities occur. This moreover depends on the masses and stiffness, for lower spring coefficients the time steps can be increased. But as was mentioned, these issues are of no concern for the method presented in section 4.5.3.

Furthermore, simulation time depends on the number of segments per initial hair strip and the used hair types. Strips are the fastest representation and strands the slowest. For the strips of a hair style with 78 base hairs and 8 segments each, with explicit Euler the simulation time was around 10 ms without collisions. The simulation of a pure cluster and strand representation took almost an order of magnitude longer, and the combined LODs (again without rendering them) needed about 30 ms on system **A**, whereas the threshold was chosen such that the proportion of strips dominated. The choice of the various LOD's geometrical representations also impacts performance: while for example clusters consisting of cylinders with more than around ten side parts appear visually more pleasing, likewise computing time increases.

In addition, with bigger tree depths performance decreases and memory consumption increases, which e.g. depend on given thresholds and the hair strips' initial widths. For 90 initial strips and a tree depth of four levels we already have more than 21,000 hair nodes. In general, around 80 to 100 initial hair strips with circa eight segments each are enough for simulating a hair style. Apart from simulation issues this is already an enormous overhead for rendering alone, as the grouped LOD representation requires traversing and evaluating a complex scene-graph hierarchy on the CPU with every render frame instead of simply updating the vertex buffer on the GPU as in our previously presented wisp-based approach, thereby leaving less processing power for other things.

4.6 Conclusions

Though character animation or speech synthesis are by no means the main research topics here, and an in-depth discussion is far beyond the scope of this thesis, appropriate techniques are necessary in the context of multimodal dialog systems to fulfill the requirements of higher control levels. This not only requires flexible control of the character and thus a flexible animation system, but also the consideration of resultant dependencies like hair movements that need to be simulated, when online motion generation and other external forces come into play. Although a lot of work already is done towards real-time simulation of deformable objects like cloth and hair, the target of research usually is conducted in a single standalone application without embedding the algorithms proposed into a wider field of applications. We hope to achieve more efficiency by integrating the proposed algorithms into existing open standards like X3D/ H-Anim [336, 335].

Figure 4.27: *Screenshots from previously described wisp-based hair simulation (above) and for comparison some photos of real long, blond hair (below).*

Hence, in this chapter we first have discussed, how humanoid animation can be more efficiently and transparently integrated into X3D to allow for dynamically coordinating the conversational behavior of virtual humans. The current H-Anim standard only defines the skeleton setup including a skins and bones system for seamless skinning. Character animation itself is mainly accomplished with timers and simple linear interpolators based on predefined animation sets. However, definition and handling of animations have never been part of the H-Anim standard, and the built-in X3D animation mechanisms are not suitable for dealing with multiple animations that shall be combined and concatenated dynamically during run-time. In addition, X3D still does not provide any support for text-to-speech and lip synchronization.

Thus, to overcome some of these limitations, we have presented a few enhancements to the present X3D standard, comprising extensions for speech synthesis and lip synchronization, as well as for the animation of characters and other objects, including the ability to mix an arbitrary number of animations of different types, by providing nodes for encapsulating and controlling animations. This will be further elaborated from a more abstract point of view in chapter 7. Moreover, we have explained the challenges of dynamics related to virtual characters, covering play-back and blending of predefined animations, as well as online motion generation and hair simulation. It is important to note that the proposed animation control component is also capable of handling dynamic motion synthesis and is thus extensible in consideration of new concepts of motion generation.

However, there are still a lot of open points left for future research such as improving the procedural animation capabilities, which are currently only implemented on a proof-of-concept level. The same goes for parameterizable motions. Though latest research shows that procedurally generated locomotion or gestures are not only very flexible but can also appear quite plausible, computation time on this level is still far beyond real-time and thus out-of-scope here. In addition, a more differentiated coarticulation scheme as e.g. described in [11] is necessary for better lip-sync.

Furthermore, in this chapter we also proposed a realistic looking method for simulating and rendering human hair in real time, applicable in complex scenarios with arbitrary moving characters. Because of the chainlike structure and some simplifications, our heuristically motivated hair simulation system is computationally cheap and properties like pliancy can be easily included. Moreover the system runs numerically very stable, is fast and robust, looks convincing, and is easily parameterizable.

Therefore, we have additionally presented our proposed scene-graph node interfaces for X3D integration. However, by exploiting modern GPU features such as stream output (i.e. writing from the vertex shader to another vertex buffer while skipping rasterization) like in the method presented by Tariq [306], performance can be further optimized. Recently, she also presented a demo utilizing the latest DirectX 11 features to directly tessellate the base hairs on the GPU, while simultaneously handling possible collisions for multi-strand interpolation [352]. But currently this requires expensive high-end graphics hardware.

By adopting approaches from physically-based rendering to simpler GPU-based methods, we are still working on a phenomenological level. Nevertheless, by including both transmission based lighting terms TT and TRT the result is very close to photorealism even for light colored hair, which is shown in Figure 4.27. Moreover on modern graphics hardware our hair shader is fast enough for the use in real-time applications. We have also presented a tool for hair styling that is realized as Cinema 4D plug-in [216], and improved usability and flexibility by embedding the simulation likewise into the plug-in. Thereby users are able to create realistic hair styles within a couple of minutes.

For further comparison we've also implemented an adaptive hierarchical simulation algorithm that is based on a mass-spring-system and emphasizes the dynamic transition from clustered hair wisps to single hair strands. But despite the inherent problems of mass-spring approaches this method exhibits the typical LOD problems and has a higher CPU load, albeit – at least for the cluster and strand levels – it can also deal with curly hair by displacing the rest angles. The main drawback of our approach is the problem, that without any further extensions the simulator is only suitable for smooth hair styles. Therefore, the possibility of extending our system for the simulation of other hair styles like curls, e.g. by following the approach of Koster et al. [191], should be investigated, though this requires more geometric detail which slows down the rendering speed.

5 Skin and Emotions

After having discussed character animation and various aspects of dynamics in general, in this chapter we first present suitable methods for real-time skin rendering [155, 156]. A closely related topic discussed afterwards is rendering of character emotions like blushing, pallor, and weeping, which occur due to psycho-physiological processes and thus can't be controlled deliberately [155, 149, 148, 162]. Additionally, we present techniques to simulate sweating and weeping in real-time [187, 148]. We also propose a parameterizable model to classify and control such manifestations of strong emotions consistently with other behavior [149, 162], as well as the outcome of an experimental study to find out whether considering them can help improving the perception of certain emotions [338, 159].

5.1 Extrinsic Factors of Skin Rendering

Simulating the visual appearance and lighting of human skin is a difficult task, which is addressed by researchers for several years. Because of the availability of high performance, programmable graphics boards, now it is possible to use techniques formerly only available to offline rendering. While the meshes for the virtual characters have to be designed and crafted in a pre-production phase, some of the phenomena appearing with skin can be computed during runtime to keep pre-production-time as short as possible.

Thus, in this section we present solutions to improve the visual quality of skin by enhancing physically and physiologically motivated techniques to real-time application and incorporating them into the X3D standard, while considering important aspects such as the composition of human skin as well as reflection properties and scattering aspects. Furthermore dynamic properties like aging and emotional changes in the appearance of a face are introduced to obtain convincing results at real-time frame rates [155, 156, 149].

5.1.1 Reflection and Scattering

As motivated in section 3.5, a couple of phenomena must be considered when rendering skin, since it consists of several inhomogeneous, semitransparent layers. Only around five percent of the incident light are directly reflected from the epidermis, whereas a thin layer of hair and moisture on the skin further impact the reflection distribution. The major amount of light travels into deeper layers of skin, where it gets scattered multiple times and therewith colored based on the particular melanin and hemoglobin concentration, which e.g. causes bright light shining reddish through the ears. For approximating subsurface scattering during runtime [106, 155, 64], the distance light travels through skin can be measured with the same technique as is used for shadow mapping (cp. Figure 5.1).

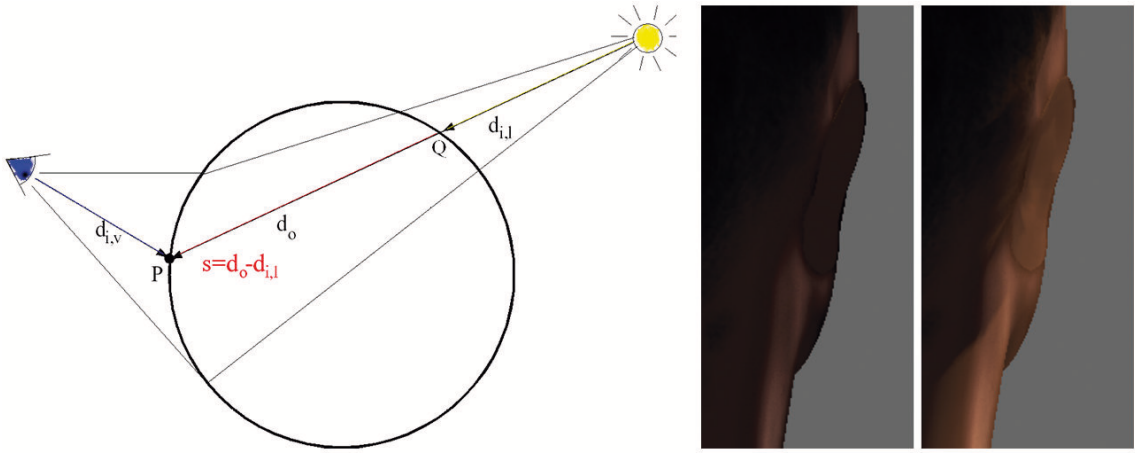

Figure 5.1: *Left: travel distance $s = |Q - P|$. Right: ear without/ with scattering effects.*

For accomplishing such more advanced rendering techniques, we first need to introduce the concept of multi-pass rendering in the context of X3D [156], which is outlined in section 6.3.4. Thereto, the standard X3D scene-graph conceptually has to be extended to a material and effects graph. For creating dynamically generated maps, e.g. an extended *RenderedTexture* node [347] can be used in order to provide the ability for off-screen rendering including associated buffers like the depth buffer.

Our modified *RenderedTexture*, whose interface can be found in section 6.3.4, has an additional SFBool field called "depthMap", which allows the automatic generation of depth maps for e.g. additional user created shadows as needed for the light pass of the skin shader (compare Figure 5.2, left) when shadowing is not enabled or higher quality is required. Otherwise, the scene shadow map can be reused.

To explain our subsurface scattering approximation method, a short summary of shadowing techniques is needed before going into detail. Shadows are not only needed for approximating scattering but they also provide a scene with depth cues. The most commonly used technique is shadow mapping, which is described in more detail in sections 3.2.4 and 6.3.3. After having rendered the depth map from light view, for each image pixel one path from the light source to the surface and its distance s between entry point Q and exit point P is determined (see Figure 5.1). Now s can be used to calculate the final color at point P per fragment by using the following equation:

$$I_{out} = G \cdot I_{in}, \; G = (d_{max} - s)/d_{max} \tag{5.1}$$

Here I_{out} is the resulting color; I_{in} is the scattering color, d_{max} the maximum length before complete absorption of light, to which s is being clamped, and G is a geometric term which relates the outgoing intensity to the material's thickness. Although in nature this relationship is not linear, it leads to good results (Figure 5.1, right) and runs fast even on older GPUs. Alternatively, an exponential fall-off function can be used. Direct scattering at P is simulated too, because for the side of light incidence we have $s = 0$.

Furthermore by encoding the distance into the color channels of preferably a floating point texture the algorithm works even without knowing the scene sizes in advance. The color

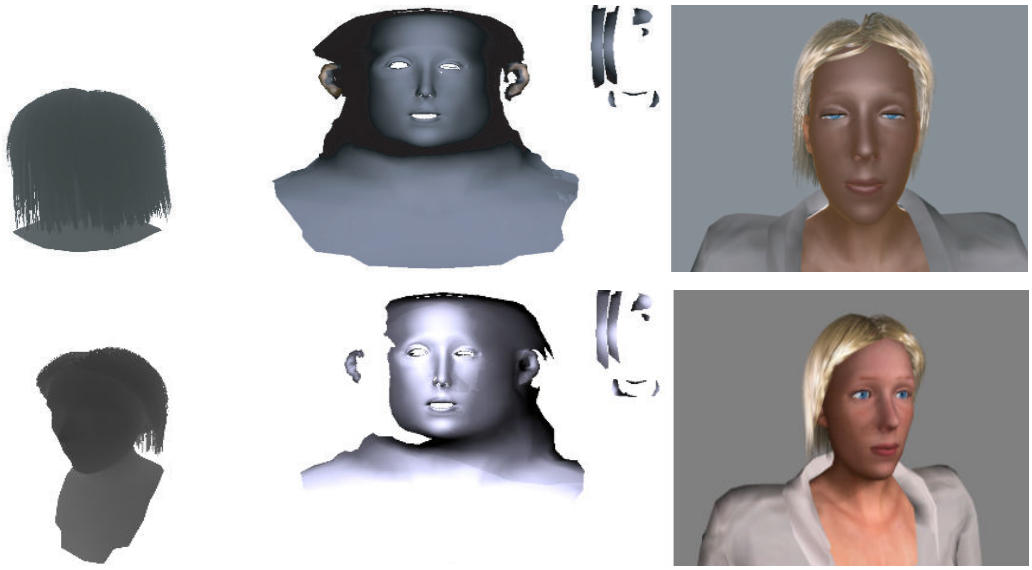

Figure 5.2: *Subsurface scattering approximation via multipass rendering (above with lighting from behind, and below the light is in front of the character): depth map resulting from light pass (left), blur pass (middle), and final render pass (right).*

I_{in} is affected by the amount of melanin in the epidermis and hemoglobin in the dermis (considering absorption and scattering effects). Finally, if light transits skin as in the case of the ears, the color given by hemoglobin is more significant then its melanin-part.

Due to the ease of this method, it is not necessary to do difficult pre-computations and the underlying geometry can be fully dynamic. Some drawbacks are that neither media transitions nor entrance and exit angles are handled (see Figure 5.1, left). Moreover, if the geometry contains wholes, this leads to rendering errors. And lastly, this raycasting-based approach is not suited for Image Based Lighting (IBL), what needs to be considered in the context of Mixed Reality applications (cf. section 6.3.2).

We further improve the scattering effect by combining this technique with the texture space rendering algorithm outlined in [269]. First the shadow map is created (Figure 5.2, left), then the diffuse illumination with shadow tests including equation 5.1 is rendered into a texture map at the position of the texture coordinates (cf. Figure 5.2, middle), and in the final pass (Figure 5.2, right) the illumination map is blurred and the resulting color is mixed with a modified Henyey-Greenstein term [125] for approximating the phase function of dermis and epidermis, which are different in their scattering properties.

This phase function is calculated in the fragment shader like follows, where the parameter g is the mean cosine of the scattered light directions [115, 138], and b denotes isotropic scattering that is derived from Rayleigh scattering, which describes the scattering properties of elements whose sizes are much smaller than the wavelength of the incident light. The fractional part mainly accounts for Mie scattering, where the size of a cellular element is close to the wavelength of light (cf. [133, p. 37]).

```
hg = b + (1. - b) * (1. - g * g) / pow((1. + g * g - 2. * g * LdotV), 1.5);
```

Sheen is especially noticeable when skin is seen from greater angles. This effect is inten-

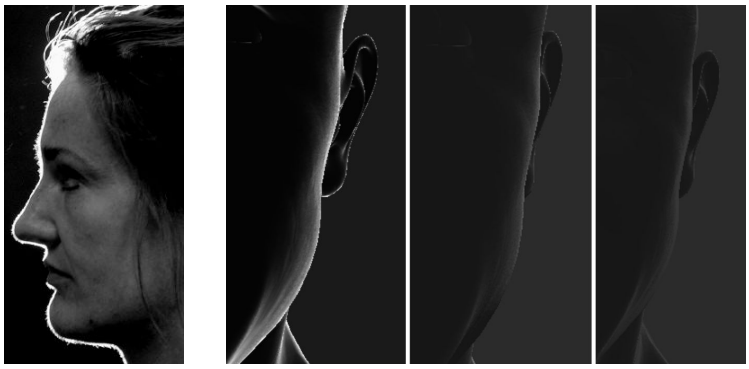

Figure 5.3: *Lighting from behind: photo (left, cf. [184]); comparison of 3 approximations.*

sified by the so-called asperity scattering shown in Figure 5.3 (left) and is caused by dead cells in the top most skin layer, tiny hairs (the so-called vellus hairs [184]), and a thin, oily layer of sweat covering the whole skin. Only about five percent of light that incidents skin is reflected directly, but this is essential for realistic imaging. Although this effect is very subtle in the final image, it is noticed when missing.

The basis for these calculations is set by Snell's law and the Fresnel equations for dielectric materials [86] such as glass, which give the highest reflectance at gracing angles, and depend on the refractive index and the angle of incidence. Because of their high complexity, a fast and reliable approximation for the Fresnel factor should be made, so generally polarization is ignored. One such approximation for determining the reflection coefficient can be found in [278], but it still has the disadvantage of a wavelength dependent term f_λ, that denotes the reflectance at normal incidence and usually is assumed to be constant with $0.1 \leq f_\lambda \leq 0.6$ due to the lack of measured values.

$$F_\lambda = f_\lambda + (1 - f_\lambda)(1 - \vec{h} \cdot \vec{l})^5 \tag{5.2}$$

The dependence to f_λ has to be eliminated, and the light and viewing directions as well as the original index of refraction η (with $\eta \approx 1.4$ for skin tissue) should also be taken into account. Therefore we propose a solution [155], which is based on [122] but is about 60% percent faster and makes furthermore use of a parameter p, which e.g. can be defined via a gloss map, to describe the moisture respectively specularity of human skin (with $\vec{n}$ = surface normal, $\vec{l}$ = light vector, and $\vec{v}$ = eye vector):

$$F_\eta = p\eta[(1 - \vec{n} \cdot \vec{v})^5 \vec{n} \cdot \vec{l} - (1 - \vec{n} \cdot \vec{l})^5] \tag{5.3}$$

Because the vellus hairs can be seen as point scatterers, the result finally is scaled with some noise. Equation 5.3, clamped to zero, can easily be computed in the fragment shader and is only about 10% slower than evaluating equation 5.2, but it therefore gives better results even in close-up views and does neither lead to too bright and metallic looking surfaces nor is it too restrained. Figure 5.3 (right) shows a comparison of the three discussed approximations – from left to right: equation 5.2 [278], our solution (equation 5.3), and the approach described in [122].

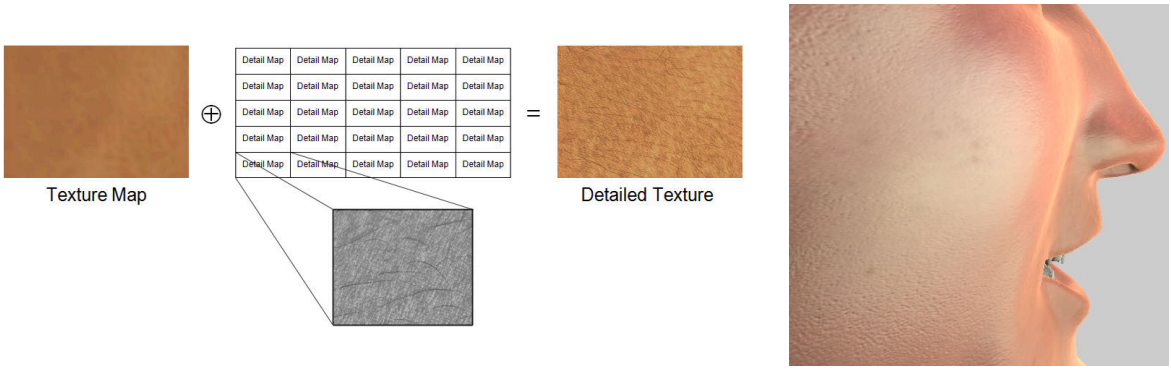

Figure 5.4: *By combining the normal texture map with a multiple tiled detail map, a texture with a high detail level is obtained.*

5.1.2 Resource Management

In combination with real-time rendering often memory resources are not considered well enough. Although, in contrast to geometry, textures seem to be computationally quite cheap here, an RGBA texture with 2048 × 2048 pixels and mip-mapping enabled requires about 21 MB of memory on the graphics card. Especially when having big scenes and older cards below 128 MB or in the area of embedded systems this can soon lead to problems. Besides using downscaled textures or various texture compression formats like DXT,[1] detail mapping allows having a high amount of detail with low memory consumption.

The idea of detail mapping is to simply split up a highly detailed texture into two layers: a smaller low resolution texture that only provides coarse characteristics and unique parts as well as a high resolution texture that contains high frequent details and no unique information. This texture is scaled down and layered in multiple tiles on top of the other texture. Therefore, the texture first has to be made tileable, so that the borders fade into the opposite side, and the low frequent image parts are removed with a high pass filter. By modulating both textures the result then looks like a high resolution texture. As shown in Figure 5.4, the detail map usually consists of a close-up photograph of the skin such that individual tiny hairs, wrinkles and pores are visible. The same principle is also applicable for normal maps (cp. Figure 5.4, right).

5.1.3 Aging

Wrinkles appear with certain facial expressions such as wrath, but also with aging, which is a slow facial signal and can change the appearance of human skin noticeably over time. Thus, the most important phenomena that happen are wrinkles, skin is getting thinner, thus appearing more translucent, and slightly turning to gray, and age spots are emerging. Well suited techniques for the visualization of wrinkles are bump maps and displacement maps. The latter now are supported with Shader Model 3.0, and can be used for real vertex displacements, but need a highly tessellated mesh.

In contrast, bump maps [30] change a model's appearance by bending its normals, and

[1] http://opengl.org/registry/specs/EXT/texture_compression_s3tc.txt

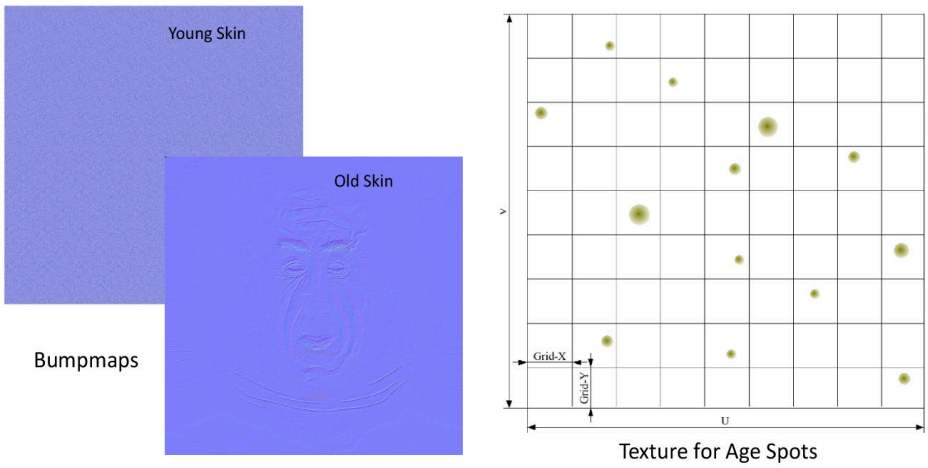

Figure 5.5: *Pores and wrinkles via bumb maps (left), and jitter texture for age spots (right).*

no real geometrical deformation is carried out. Nevertheless, bump maps create a good illusion even for wrinkles, where age heavily has left its trace. Though there exist more elaborate methods for simulating aging based on a database of 3D scanned faces on which learning is applied [274], the costs in terms of example face data and computing time are too high for our use case. Suitable bump maps contrariwise can be easily created with many DCC tools out of the box and smoothly fit into prevailing content creation pipelines. Consequently, we stick to the latter to simulate the emergence of wrinkles. Therefor, at least two bump maps are needed: one for the young and one for the old skin (Figure 5.5, left). These maps then have to be interpolated. Although wrinkles are not proportional to age, for most cases a linear interpolation creates good results.

When using the texture layer method described in the next section, the interpolation can be fully done by the hardware. This method is also suitable for creating expressive wrinkles resulting from muscle movement. As explained in section 3.2.2, bump mapping usually is carried out in tangent space. We therefore have extended the X3D *TextureCoordinateGenerator* node with an additional value "TANGENT" for parameterizing the 'mode' field appropriately [279]. The MFFloat field 'parameter' thereto contains optional mode parameters that e.g. for tangent calculation could be $[0, 1, 2]$, which means that the source texture coordinates used for calculation are in slot 0, and the calculated tangents and binormals shall be in the texture coordinate slots 1 and 2. With the matrix $M = \left(\vec{T}\ \vec{B}\ \vec{N}\right)^T$ all vectors can then be transformed from object to tangent space.

Not only hair is changing color through aging, skin also turns slightly to gray. As the epidermis is getting thinner with age and blood cells as well as the number of melanocytes are decreasing, its color is changing through the years. This can be simulated by diminishing the variance of the RGB color triple by equally weighting all color channels depending on the desired age. With I_{rgb} being the original skin color, O_{rgb} the grayed color, and $c \in [0, 1]$ a suitably chosen interpolation parameter for determining the age, the resulting equation for smoothing the original skin color thereby is:

$$O_{rgb} = c \cdot I_{rgb} + (1 - c) \cdot (I_r + I_g + I_b)/3 \tag{5.4}$$

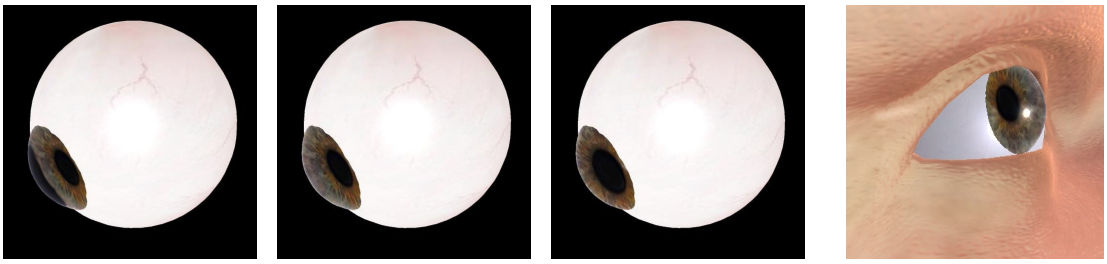

Figure 5.6: *Left/ middle: rendering of iris with different indices of refraction. Right: specular highlight and ray-casting in fragment shader for simulating refraction.*

Age spots, only a few millimeters up to centimeters in size are pigment crowds in the upper skin layer. They are set up both for men and women and are observable signs of long lasting sun exposure. On areas, which are exposed to sun frequently, like face or hands, they appear more often, but still randomly distributed. With age, they appear in higher amount and scale. For visualizing this phenomenon, a technique is chosen that is known as jitter texture (see Figure 5.5, right).

Similar to jittered sampling [285], the texture is split virtually in grids, and in random grid fields a spot is drawn with random position and size within given boundaries. For simplicity, a quadratic texture with size $U = V = 2^n$ is chosen. To simulate the qualitative and quantitative growth of age spots, the spots are drawn with different alpha values that are taken into account when blending the texture with the skin. Thus there is no age spot noticeable until a certain age. Since the appearance of age spots varies from one person to another, this boundary has to be chosen individually.

5.1.4 Eyes

The human eye constitutes an important part of the non-verbal communication. Hence, quite as essential as natural looking skin are realistic eyes. Yet usually only a character's view direction is controlled by means of a special gaze controller. Therefore, typically procedural animation methods as discussed in section 4.3.3 are used. For simpler setups, where it is only necessary that the character always looks at the user (cp. Figure 4.17), an angularly restricted billboard may suffice.

Apart from plausible motions, for extreme close-ups also a natural appearance is essential. But until now, if it all, this topic mostly was treated rather artistically. Aiming at a anatomically correct rendering of the human eye, François et al. [87] recently presented a method to unrefract iris photographs, which in turn are used as input textures for their real-time rendering method based on their proposed refraction function. Similarly, an earlier NVidia Cg demo[2] utilized similar rendering techniques.

Likewise, we also account for refractions. Thereto, we use two different GLSL shader programs for rendering the eye [156], one for the eye ball and the other for the iris. The main task of the eye ball shader lies in the realistic synthesis of the specular highlight on the vitreous. Additionally, an environment map is applied in both shaders. The shader

[2]GLSL version: http://developer.download.nvidia.com/SDK/9.5/Samples/samples.html#glsl_eye

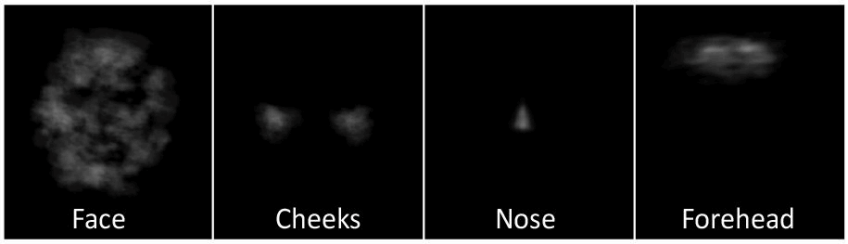

Figure 5.7: *Exemplary maps for denoting different facial regions.*

for the iris additionally has to handle the refraction of light passing through the cornea.

Therefore, the shader has parameters for iris size and the index of refraction of the cornea. Refraction itself is done in the locale coordinate system of the sphere like iris geometry by putting a virtual plane orthogonally to the avatar's viewing direction. Just as with ray-tracing the refraction vector of the "viewing ray" is calculated. If no reflection occurs, the intersection with the virtual plane is computed and used as view dependent texture coordinates for the iris texture (cf. Figure 5.6).

Another aspect not yet mentioned are emotions. Whereas behavior like rolling one's eyes or being wide-eyed of fear again can be modeled with a gaze controller or via mesh deformations, the eyes also get a reddish coloration when being in rage or grief. This can be achieved by e.g. additionally interpolating between two different eye ball textures as will be further outlined in section 5.2.1 for facial skin.

5.2 Rendering Emotions

As was shortly touched in the previous paragraph, skin appearance is not static but can change due to emotions and the like, a fact that usually is ignored in real-time rendering. Hence, in this section we will concentrate on visualizing human emotions – especially emotionally caused skin changes like blushing, paling or even crying, since these can be an essential aspect of non-verbal communication, too [155, 156, 149, 162, 338].

Although dynamic skin changes cannot be controlled deliberately by a human being, these intrinsic factors of skin rendering still need to be controllable and thereby scriptable by the application in order to be consistent with other expressions and to possibly intensify them. Therefore, we present our emotional model, which we have used as basis for animating changes, as well as the outcome of an experimental study to find out whether considering such skin changes can help improving the perception of certain emotions.

5.2.1 Facial Coloration

In this section solutions for skin turning red or pale are discussed [155, 156]. This usually comes along with exertion and strong emotions. For some emotions, changing of skin color is remarkable and caused by varying blood circulation. When feeling shame or embarrassment, cheeks, ears, nose and forehead are blushing, and when being sick or feeling disgust or fear, the face gets pale (compare section 3.5.2). Moreover, for many

Figure 5.8: *Texture stacks for face color (left) and additional normal maps for crying animation.*

emotions wrinkles can occur due to muscle movements, e.g. for joy, there are small wrinkles around the eyes, which can be displayed with the help of bump mapping using similar methods as described in section 5.1.3.

Since the color variation of emotionally caused skin ton changes can be limited to several face areas, a texture map that denotes these areas seems adequate. The artistically creation of this map in preparation of the final scene is an obvious drawback of this method. But as already stated, many aspects concerning emotions like blood flow and changes in coloration still remain unexplored. Keeping also in mind, that all rendered geometry has to be created in advance, it is not possible at the moment to find areas affected by varying blood circulation fully automatically.

There are basically two possibilities for creating the maps [155]. In a naïve first approach one can think of embedding some ranges like nose, cheeks, forehead and the whole face as grayscale images within the four channels of a RGBA texture map (see Figure 5.7). But this way the computations in the shader program need special treatment, get expensive, the original color information gets lost, and the result looks quite unrealistic especially when trying to synthesize pale or very reddish faces during runtime.

The second approach uses a 3D texture containing all color information (Figure 5.8, left): just like a stack of plates the pre-designed 2D images are layered within a 3D texture, starting with the palest face image, over neutral, and ending with the reddest face. This method prevents setting invalid emotional stages and also leads to much better and faster results by employing texture fetching hardware for color interpolation.

This is automatically done by assigning each main emotional image a number between zero and one (e.g. 0.75 for anger), which then is used as the z-coordinate for indexing into the 3D texture image stack. Figure 5.8 shows an example, where each 2D texture makes up a layer in the 3D texture. As is shown to the right, the same technique can be used for normal maps (in this case to animate tears). By changing the third texture coordinate continuously from 0 to 1 a smooth color animation is achieved.

Then the resulting color value is taken as the base color of the skin shader explained in section 5.1.1 or alternatively directly as diffuse color. This way, no additional shader programs are required and skin color changes can be integrated easily into existing skin rendering methods that also consider reflection properties and scattering. Likewise, wrin-

Emotion	Changes in/ on facial skin
Neutral	Neutral face color, no changes
Joy	Rosy cheeks
Enthusiasm/ Ecstasy	Rosy cheeks, tears of joy
Surprise	Rosy cheeks
Disgust	Pale cheeks
Down	Low lacrimation
Sadness	Blushing cheeks, raised lacrimation
Grief	Blushing cheeks, red blotches, intensive lacrimation
Apprehension	Pale cheeks
Fear	Pale cheeks, then pale in whole face
Panic	Pale face, low lacrimation, sweat on forehead
Annoyance	Blushing cheeks
Anger	Blushing cheeks, red blotches in the face
Rage	Blushing cheeks, red blotches in face, finally red face, sometimes tears

Table 5.1: *Overview of visually distinguishable emotional states caused by vegetative functions, used for the parameterization of emotions that result in different facial complexions.*

kles can analogously be animated by interpolating between bump maps.

5.2.2 Emotional Model for Classification and Parameterization

In this section, we outline the emotional model we have used as the basis for generating skin changes caused by emotions [149, 162] and an evaluation of the results [338, 159]. Emotion simulation systems not only require control over all modalities for achieving consistent behavior but before being able to display them, they also categorize the calculated continuous emotions into several discrete emotions. Albeit emotion simulation itself is out of scope here, we present a model for dynamically parameterizing emotional skin tone changes like blushing, turning pale and similar symptoms according to existing methods for simulating emotions. It allows embedding these changes into a well defined parameter space for being consistent with posture and mimics.

For creation and composition of such emotion data, we have developed a special emotional model, which takes all visually perceivable vegetative symptoms of different emotions, such as facial coloring, into account. These can be caused physically or emotionally, but in both cases are controlled by the ANS [273]. Our parameterization and classification model (see Figure 5.9) is based on physiological knowledge (cf. e.g. Shearn et al. [284] or [211]) as well as on the previously mentioned basic emotions of Ekman [77] and the psycho-evolutionary emotion theory of Plutchik [249]. But in contrast to Plutchik's model our model is only two-dimensional. In addition to that, it belongs to the class of decoding models, since for this purpose it was specially adopted to represent parameterizations of emotions that result in different skin tone changes and so on. The proposed differentiation of visually distinguishable strong emotions is shown in Table 5.1.

For plausible dialog behavior, emotionally caused skin changes need to be customizable

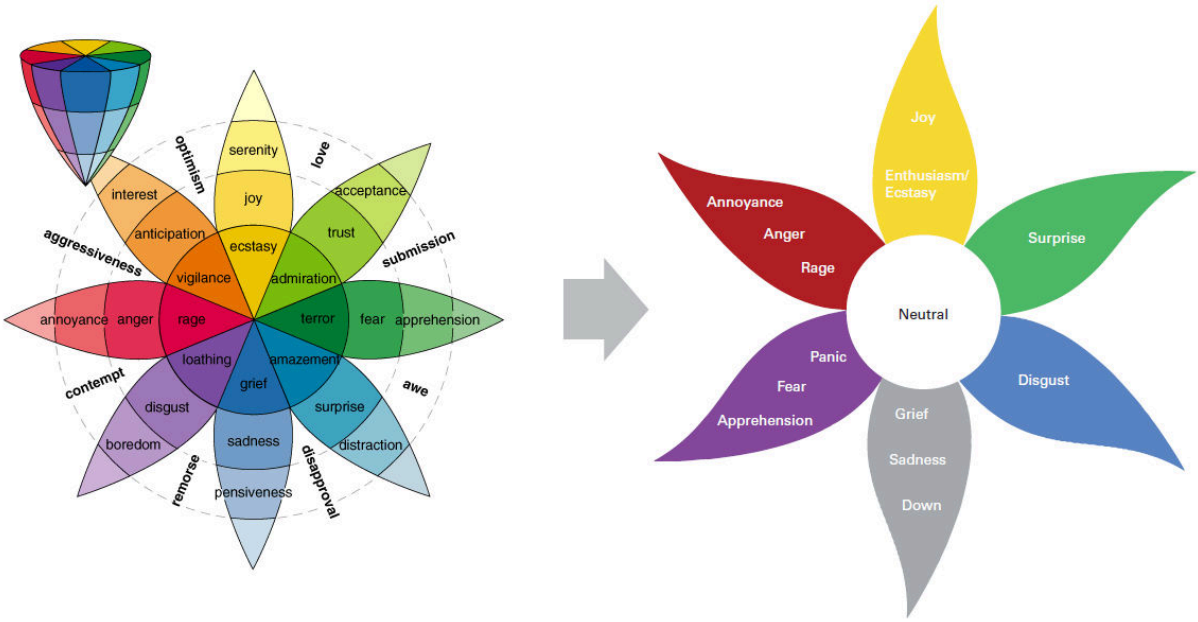

Figure 5.9: *Visualization of our derived emotion model (right, cf. Table 5.1) that is based on Plutchik's psycho-evolutionary emotion theory (left).*

and consistent with existing animations. Therefore, we need to know whether the face should blush or get pale, in which area the color should be changed, and the intensity i and duration Δt of the changes. The latter defaults to the average value of $\Delta t = 35s$. Color changes are achieved by interpolating between accordingly colored face textures (starting from a pale face up to a completely red face) stacked in a 3D texture for having hardware acceleration as was described in section 5.2.1, which yields a simplified integration into existing skin rendering methods.

In general, besides skin changes it is also necessary to synchronously handle corresponding changes like wrinkles around mouth and eyes, as people mostly focus on these regions. For simulating crying in real-time (see Figure 5.12), additionally a GPU-based on-surface droplet flow simulation was developed (as described in section 5.3), with some extensions to handle non-planar surfaces such as faces, as well as different contact angles, which is especially of interest when simulating bigger beads of perspiration. Thereby, tears and beads of perspiration are integrated by specifying drop sizes and droplet sources in image space via texture coordinates and by starting the droplet flow simulation.

As mentioned in the beginning, these signs of strong emotionality need to be controllable the same way as other character motions for being consistent with the avatar's posture and facial expressions. Therefor, we have integrated the possibility to also control skin changes into our behavior control framework (see section 7.3). Following the concepts of X3D (that only provides lightweight components for real-time 3D graphics content embedded in applications – but not the applications itself), the corresponding building blocks, or X3D nodes respectively, have no further semantics connected to them.

For example, there is no notion of an "intelligent" object named *Eye* that knows when to blink and how to cry. Instead, at the X3D level the nodes can serve many purposes

Figure 5.10: *Different textures for texturing the faces of virtual characters, which were shipped together with the geometric character models.*

depending on the application. Thus, for being used in the control layer, first the droplet flow simulation as well as the skin shader together with the z-index need to be wrapped by "AnimationContainer" nodes and linked with the corresponding animation script (cp. sections 4.3 and 7.3.4). Then, everything can be controlled and animated in a unified manner at a higher level. An example script can be found in section 7.2.3 on page 210.

5.2.2.1 Generating Emotion Data

Emotionally caused skin changes should be customizable and based on existing expressions. For the change of color the individuality of the models must be taken into account, since each facial texture is different concerning its texture coordinate mapping, color and size, as shown in Figure 5.10. Thus, a consistent color for blushing and pallor in most cases cannot be used. In addition to the colors, the areas for color changes have to be defined. Just like the colors, the areas often must be determined individually.

Hence, individual character data for each model is generated in a pre-processing step, and can later be used for animating facial color changes. For the realization of blushing and pallor, the following information is important:
(i) The logical color c, i.e. the information whether the face should blush or get pale;
(ii) the logical area f in which the color should be changed (e.g. cheeks or forehead);
(iii) the intensity i and duration Δt.

With these three pieces of information the tears and beads of perspiration are not yet considered. This data is separately defined, because on the one hand it does not occur for every emotion, and on the other hand, it is integrated a bit different into the framework, namely by specifying the drops sizes and the droplet sources in image space via the texture coordinates and starting the on-surface droplet flow simulation as previously described.

To visualize emotionally caused complexions in the avatar's skin, only texture data should be taken into account because of the fact that various skin shaders are often used for rendering virtual characters. Therefore, a separate shader for the visualization of the color changes is not appropriate. Hence, to automate the work flow regarding content creation, areas like the cheeks that will change their color must be identified first. One possible approach is to use computer vision techniques as e.g. provided by OpenCV [230]. This library includes methods for pattern matching, which can be used to create distinctive areas such as the eyes, nose and mouth on any facial texture.

But because of the different parameterizations of possible textures (see Figure 5.10) the recognition of distinctive areas without considering the whole geometric information is at

Figure 5.11: *Resulting face area from the defined feature points (analogously to the MPEG-4 parameters, left) for the cheeks, which is shown exaggerated in red.*

least difficult, and for some texture types (e.g. for comic-like figures) impossible. While for example in one texture the eyes can be very easily identified, it is very difficult to recognize them in another texture, depending on how different the textures are. A major problem are often the ears and nose – as shown in Figure 5.10, where the first texture contains none. The use of a uniform template is therefore very difficult. Since a stable algorithm is not feasible here, a different approach to create the textures has to be considered.

In a second approach, the areas could be found due to markers – similar to the Feature Points as defined in the MPEG-4 Facial Animation Standard [239] for morphing the geometry (cp. Figure 2.1, right, on page 40). Here, each texture is marked with feature points for eyes, mouth, and nose. The idea is to connect the points in order to determine the cheek region, but the result often is still very far from the cheek region (see Figure 5.11, left), even by using higher order curves.

Because of non-linearities in the texture mapping, distortions are introduced this way. To be able to determine the cheek region, the feature points must be adapted to the specific model. This excludes a unified method [149]. Since the textures are obviously too different, individual areas have to be determined manually anyway. Likewise, yet not for modeling blushing but for controlling crying, in [316] two additional FAP parameters were added, so that an animator can control the particles that make up the tears, but this approach requires having an MPEG-4 conformant head model.

Last but not least, the simplest and most effective method for identifying certain skin regions is the use of manually created mask textures (similar to Figure 5.7). Areas that should change their color during runtime have to be masked first. The mask textures are created for each face model and stored within the individual character data. The artistically creation of these map in preparation of the final scene is an obvious drawback of this method. But since is not possible to find areas affected by varying blood circulation fully automatic at runtime, this semi-automatic generation process is preferred.

5.2.2.2 Synthesis of Emotion and Character Data

Similar to other animations also for dynamic changes of facial coloration the character model needs to be setup appropriately beforehand. Due to the mentioned problems concerning automatizing the whole work flow at runtime, we have implemented two tools for

creating all necessary data in advance. The first module creates generic emotions data, which together with a given character model including its textures is used as input for the second module that generates specific skin textures and scripts for easing the setup process. This distinction is drawn due to the approach to somehow separate and thus unify the more abstract emotional data from the concrete character data itself.

The emotion data, i.e. the model independent logical color c, area f, duration Δt, intensity i as well as the information regarding when to start which type of droplet flow simulation, is separated from the character data consisting of individual data like the – manually created or however generated – mask textures. The first component creates the emotion data (i.e. the action of displaying emotions) by creating generic keys for every event. The second component then assigns the emotion data to the concrete character data, thereby combining everything to an individual emotion.

Besides the geometric model the individual character data consists of the face texture, the mask texture, the individual color and the texture coordinates for the tears. By first separating both types of data before synthesizing them, we are also able to simulate e.g. comic-style avatars with non-realistic coloring and textures etc. Based on this model, emotions are selected and compiled by using the first tool. For each emotion a standard duration of $\Delta t = 35$ seconds is defined. This time can be adapted to an individual animation. Then, similar to a config file, generic keyframes are generated for the selected emotions with the help of the first tool, containing at least c, f, and Δt.

The second tool requires the generic emotion data, which is then used to be merged with the character data in order to produce a change of the face color or to be able to trigger a perspire or crying simulation. The output consists of the 3D texture for the face coloring as well as the corresponding X3D and animation description files for describing the time-based behavior of all emotion data. The latter will be shortly explained in section 7.2.3 with the help of an example. Though special tools are not necessarily needed here, they ease the process and should be integrated into modeling packages such as 3ds Max, which already allows creating normal maps along with facial expressions.

For creating the final emotion data we have compared two methods: similarly to a keyframe animation the first encodes all emotions into one 3D texture in advance, whereas a z-offset and length determine a particular emotion (e.g. joy between 0.25 and 0.4). Besides the ease of use this has the advantage that rosy cheeks, red spots etc. can be parameterized differentiated, but the disadvantages are high memory consumption, slow loading time, and missing flexibility during runtime. The second and preferred method was illustrated in section 5.2.1 and just stacks all textures in a 3D texture by starting with the palest face and ending with the reddest face image, since it nicely fits with the emotion classes described in Table 5.1 and does not suffer from the aforementioned issues.

Figure 5.12 (top), shows the changes in the emotional state *Anger*, starting with *Annoyance* and ending up with *Rage*. The cheeks blush and then red blotches appear. In emotion *Sadness* when weeping the cheeks are blushing too, which is shown in Figure 5.12 (bottom). By changing the texture coordinates and the color of the tears, it is also possible to simulate other effects like nose bleeding or perspiration. As can be seen, by combining standard facial expressions with facial complexions and similar effects, the corresponding emotion is not only much easier perceivable but also more plausible.

Shown images	All participants	Female	Male	Ages 7–9	Ages 11–16	Ages 21–65
Without skin changes	52.52 %	55.66 %	48.50 %	48.32 %	55.15 %	57.14 %
With skin changes	67.29 %	66.52 %	68.29 %	65.38 %	71.01 %	66.33 %
All images	61.92 %	62.57 %	61.09 %	59.18 %	65.24 %	62.99 %

Table 5.2: *Avg. recognition rates of shown emotions split into different age groups and genders.*

5.2.3 Evaluation and Discussion of Results

In this section we present the outcome of an experimental study to find out whether considering skin changes can help improving perception of certain emotions [338, 159]. Thus, the main intention of our study was to find out if facial expressions with color changes etc., like blushing in anger, are better recognized than expressions without considering physiological effects. Furthermore, we were interested if this depends on age and gender on the one hand, and on the other hand, which emotion categories benefit most of it.

Another goal was to determine how the results can be displayed more credible, and whether the resulting skin changes correspond to reality or if it should be further investigated whether certain facial regions need to be modified in color or intensity for achieving better results. To determine how well the shown emotions are perceived, the participants had to fill out a questionnaire, in which different facial expressions of three virtual characters (a female and two males) were shown.

Because not for all persons a computer was available, temporal changes such as lacrimation or change in face coloring could not be displayed, which also hindered the recognition of highly context dependent emotions. As basis of the study, a questionnaire was created containing 44 static images of emotions in a context free situation. Every image shows a facial expression of one of the characters either with or without color changes. Depending on what emotions, as classified in Table 5.1, the participant can recognize best, he/ she can select one or more emotions, including a neutral facial expression.

Altogether 57 persons (32 female and 25 male) have participated, which can be coarsely classified into three groups of different ages and backgrounds:

- 26 elementary school pupils aged 7 to 9 years,
- 17 students of secondary I aged 11 to 16 years,
- 14 adults aged 21 to 65 years (social, technical and office jobs).

The average recognition rate of all 44 images for all 57 participants was 61.92% (see Table 5.2). Images without the consideration of physiological skin changes were identified in only 52.52%, whereas images including them were identified with 67.29%. Altogether, the average recognition rate of all images for all females was 62.57%, and for all males it was 61.09%. The average recognition without skin changes was 55.66% for the females, and 48.50% for the male participants. Obviously the addition of physiological characteristics improved the perception of emotions compared to images without additional changes. Especially the male participants showed an improvement of around 20%.

The 11 to 16 year old participants did assign the images best with 65.24%, with an increase from 55% without additional skin changes to 71% with physiological skin changes. The 7 to 9 year old participants could recognize the pictures with 59.18%, with an increase from

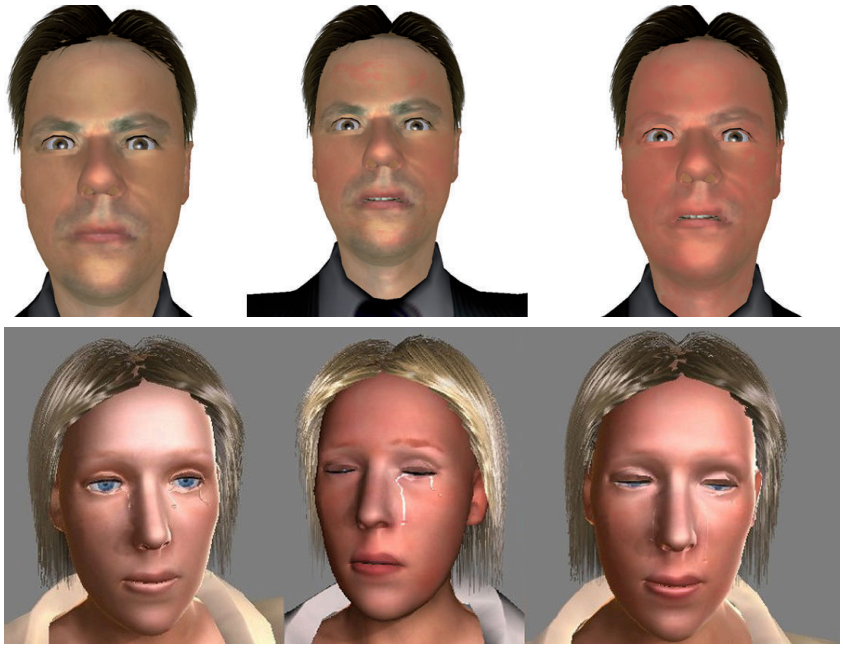

Figure 5.12: *Color changes in emotion 'Anger': first the cheeks blush, then red spots appear, finally the head gets red (top). When weeping in grief the face gets red, too (below).*

about 48% to 65%. The 21 to 65 year old participants did recognize the pictures with a detection rate of 62.99%.

Classified according to different types of emotions we have the following results: At worst the two emotions *Surprise* and *Apprehension* have been identified (on average 26%). These emotions were not better recognized by including color changes. But in this case only slight color changes are visible, which for some of the static expressions shown in the questionnaire were hardly distinguishable. This corresponds with the finding that the neutral facial expression as control condition was detected in only 52.63%. But nevertheless this result confirms with psychology, which states that those emotional classes on the one hand are not recognized well in real life either and on the other hand are highly context dependent (cf. [113]).

The emotion class *Joy/ Enthusiasm* was recognized best (94.99%, without a significant improvement by including the skin changes), probably because of the slightly exaggerated smile expression. This is followed by the emotion *Down/ Sadness/ Grief* (84.39%). Here, by especially adding the tears, we achieved an improvement from 79.53% to 86.47%. The emotional class *Annoyance/ Anger/ Rage* was recognized with a rate of 38.60% without color changes and with color changes it was 69.59%, which was the most significant improvement, probably due to the proverbial red head (see Figure 5.12, top).

Thus, emotions that additionally show dynamic skin changes in general were recognized better by each gender and age group as expected. Though the emotions of the female avatar were slightly better recognized than those of the male ones, the study has shown that the allowance of psychological and physiological effects leads to much better perception. Albeit there are still some open questions such as which psychological factors are really relevant here, which is left for future work, the outcome of our study is similar to

Figure 5.13: *Left: A slightly weeping woman behind a window pane with rain-drops running down (updated by our simulation), refracting the scene behind them. Middle: Three frames showing a crying character – the flow direction changes in real-time according to the head movements. Right: Varying simulation parameters like color and viscosity also allows modeling other types of fluids like viscous slime.*

the results recently obtained by Pan et al. [238] and de Melo and Gratch [57], suggesting that physiological manifestations especially can be used to convey intensity of emotions.

5.3 Blood, Sweat, and Tears

This section presents techniques for animating the flow of liquid droplets on curved surfaces, which work in image-space and are suitable for simulating e.g. sweat, tears, or rain drops within highly dynamic virtual environments [155, 187, 148].

5.3.1 Motivation

For several emotions simulating the looming of sweat and tears is important for correct perception or can at least increase realism. Some of those emotions can be intense anger or fear, and also exertion. For the visualization of sweat or tears it can be necessary to create e.g. meshes during runtime, thus it is not only a general surface or lighting phenomenon. It is getting even more complicated, when simulating the behavior: drops are merging and building larger drops if they collide.

As can be seen in the example shown in the middle of Figure 5.13, if the character's head moves, the tear drops also have to follow accordingly with respect to all acting forces. Because it is impossible to animate all possibilities in advance, the droplet flow has to be simulated during runtime. Based on the observations outlined in section 3.1.2 on page 66 and the techniques for real-time on-surface flows presented by El Hajjar et al. [112], with some extensions for handling e.g. non-planar surfaces and different contact angles, in this section we present a GPU accelerated approach for animating droplet flow on almost arbitrary surfaces, which we also have embedded into X3D [187, 148].

The droplet flow is advanced in time according to external forces like wind and gravity. Because the presented algorithm runs entirely on the GPU, all fluid information is held in textures that contain the current velocity and fluid volume. The texels determine, how much fluid remains and how much is provided from neighboring texels. Thus, the number of drops does not influence the performance and real-time framerates are achieved.

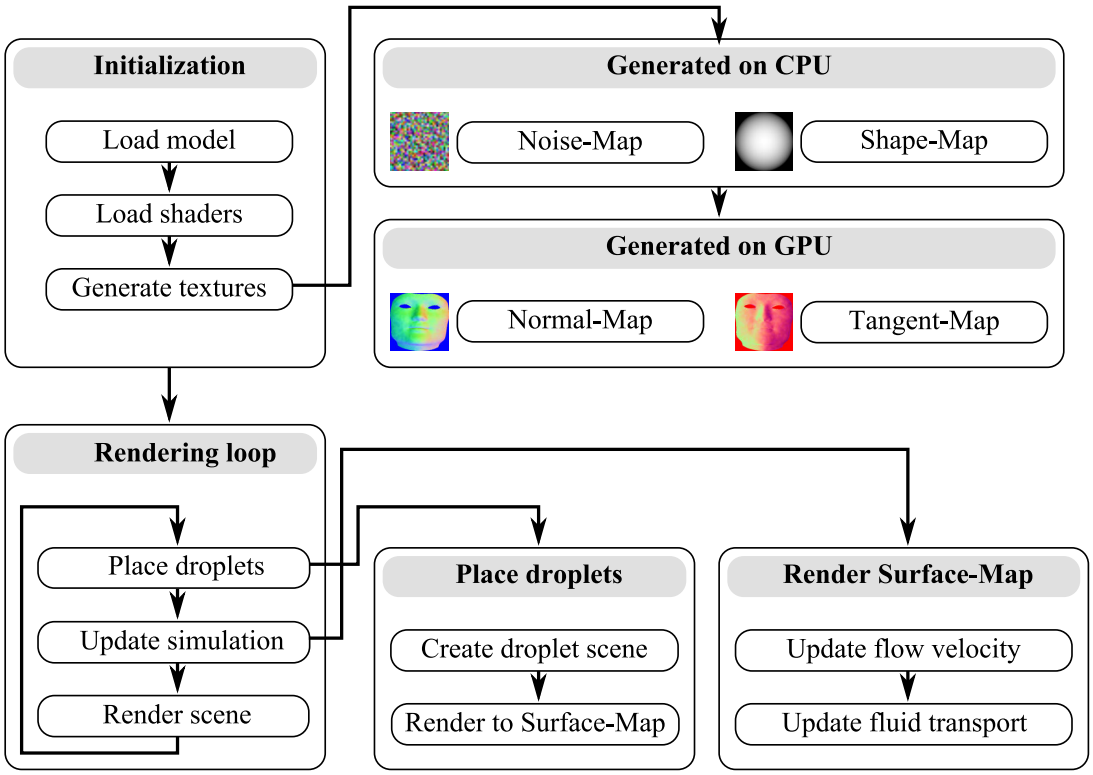

Figure 5.14: *Schematic showing the data flow of the simulation. After initialization (upper part), the system is updated before rendering the scene (lower part of figure).*

Moreover, an important advantage of this image-based approach is that collisions are handled implicitly and hence come for free.

5.3.2 Droplet Flow Simulation

5.3.2.1 Initialization

Our proposed method for simulating and rendering on-surface droplet flow is real-time capable, runs completely in image space and utilizes ping-pong rendering for evolving the flow in time. For exploiting the highly parallel nature of modern graphics hardware, all relevant data is kept in texture memory and processed by GLSL shader programs (cf. Figure 5.14 for a schematic representation), whereas the fragment shader acts as kernel.

Therefore, we make use of the chart $\Phi : U \rightarrow [0,1]$, given by the vertex to texture coordinate mapping, with $U \subset \mathbb{R}^3$ being the surface of the 3D geometry. Because there are discontinuities at the boundary $\{0,1\}$, the simulation results are only valid for the interior $(0,1)$. Furthermore, it is important that the mapping is a homeomorphism for enabling a smooth simulation in image space, that is consistent with its rendering in object space. Thus, for describing the droplet flow we use a texture called surface map. Each texel maps to a certain part of the surface, for which it describes the velocity and volume of its corresponding fluid particle (cf. Figure 5.15 (iii)).

In the upper part of Figure 5.14, the initialization of the simulation system is depicted.

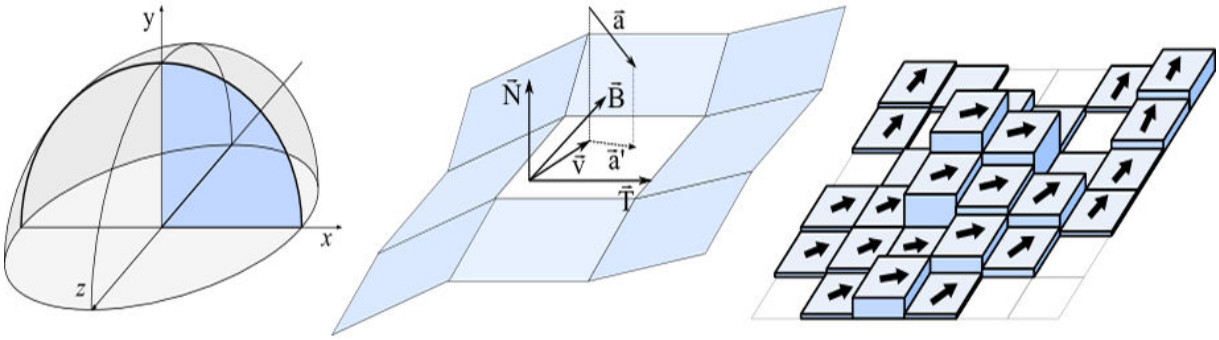

Figure 5.15: *(i) Droplet shape according to* f(*r*)*, given in Shape-Map for speed-up of droplet placement. (ii) Force and velocity calculation in tangent space. (iii) Flow velocity* $\vec{v}$ *and fluid volume q of fluid particles in Surface-Map.*

First, the 3D model and the necessary shaders have to be loaded, and then all required textures are generated. The latter usually are not changed during the simulation, and as shown in the figure, they consist of the noise, normal, tangent, and shape map. But when used in combination with bigger vertex displacements caused by morphing normal and tangent maps need to be regenerated.

Normal and tangent map are not only needed for being independent from the concrete geometry, but also to be able to handle obstacles and non-planar surfaces, too, which are not treated by [112]. Because both maps are created by using the 2D texture coordinates for rendering instead of the 3D vertices, it is indispensable, that the uv-mapping is one-to-one. The noise map is used for adding additional surface roughness when creating both other maps to account for meandering fluid flow.

Because of surface tension, drops form a dome-shaped minimal surface (Figure 5.15 (i)). As opposed to the cylindrical ones used in [112], the shape of a drop (and thus the amount of fluid), which is defined by the shape map for speed-up, depends on its radius $r \in [0, 1]$. Depending on the contact angle $\alpha < 90°$, we introduce the function f(r), described in more detail in [187], which returns the droplet height h for creating the droplet shape map. This function is thereby calculated as follows:

$$\mathrm{f}(r) = \frac{k + \sqrt{1 + (k^2 - 1)\, x^2}}{\sqrt{1 - k^2}} \quad \text{with} \quad k = -\frac{1}{\sqrt{1 + \tan^2 \alpha}} \tag{5.5}$$

The usage of user defined shape maps or pre-defined normal and bump maps is likewise possible. However, the interface (see section 5.3.4) does not yet support to set them externally to ensure an intuitive and less error-prone usage.

5.3.2.2 Droplet Placement

The lower part of Figure 5.14 shows the simulation and rendering loop during runtime. By rendering the droplet shape texture (which encodes the fluid volume, the desired contact angle, and optionally an initial velocity) with additive blending enabled (for adding the

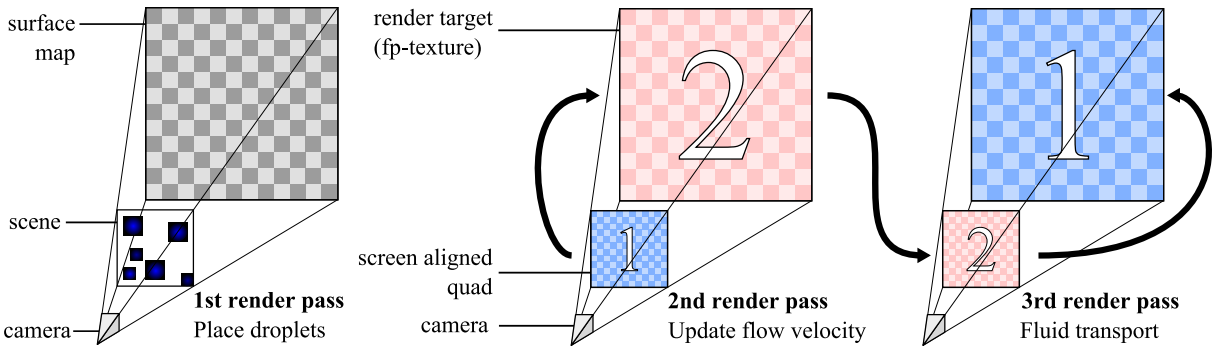

Figure 5.16: *The simulation is realized in image space via ping-pong rendering. After placing the drops (i), the system is advanced in time by first updating the flow velocity and then the fluid transport (ii).*

drops' fluid amount to the already present amount of fluid) into the surface map (Figure 5.15 (iii) shows its layout, and Figure 5.17 and 5.21 (iii) show, how it can look like), a lot of drops can be generated at once very efficiently. Because number, size and position of the placed drops depend on the type of fluid – whereas rain-drops appear randomly almost everywhere, tears only appear below the eyes – the application is responsible for when and where a droplet is placed.

Because we are using OpenSG 1.8x [232], which does not support multi-pass rendering natively, we have designed an additional `osg::DropletFlowViewport` for handling all multi-pass rendering steps, as depicted in Figure 5.16, right. Its implementation is based on the multi-pass rendering techniques as described in [157]. For each required texture or rendering pass respectively, the viewport internally holds a framebuffer object [9]. Backbuffer rendering is done with 16 bit floating point accuracy (with `GL_RGBA16F` as internal format). For one thing, this precision is enough for achieving correct simulation results, and for another thing, in contrast to fp32 accuracy it saves texture memory and still works properly on older Shader Model 3.0 graphics cards.

5.3.2.3 Update of Flow Velocity and Fluid Transport

For calculating the next time step, two sub-steps are necessary, which are depicted in Figure 5.16 (ii). First, the flow velocity is updated with respect to all acting forces. And then, according to the new velocities, the fluid transport takes place. Therefore, we render a window sized, view aligned quad into the surface map with the appropriate shaders enabled. Figure 5.17 visualizes the channels of this map for the terrain object shown in Figure 5.18 (left).

So first, for each time step Δt, the new velocity $\vec{v'}$ is obtained by integrating the following (two-dimensional) kinematic equation: $\vec{v'} = \vec{v} + \Delta t \cdot \vec{a}_p$. Here we employ the explicit Euler scheme, which despite its drawbacks is efficiently to implement within a shader program and stable enough for small time steps and non-stiff systems as in our case.

The acceleration $\vec{a}_p$ is derived from the sum of all external forces $\vec{F}$ according to Newton's second law $\vec{F} = m \cdot \vec{a}$, and by considering the relationship $m = \rho \cdot q$ for homogeneous

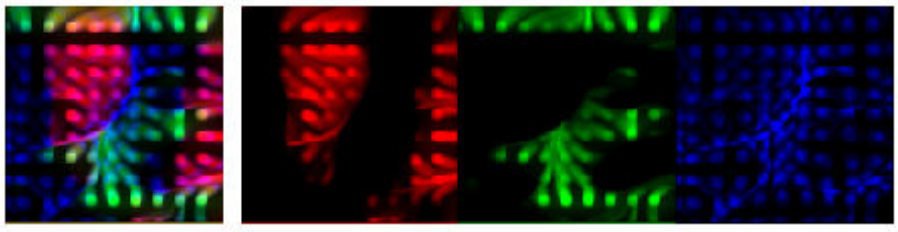

Figure 5.17: *The color channels of the surface map (left) contain the components of the flow velocity (v_x, v_y) (middle) and the fluid volume (in the texture's blue channel, right).*

Figure 5.18: *Droplet simulation applied to a simple terrain model (showing the chronological sequence of the simulation under influence of gravity), and a wet umbrella.*

materials (with mass m, volume q, and density ρ), taking into account possible obstacles:

$$\vec{a} = \frac{\vec{F}}{m} = \frac{\vec{F}}{\rho \cdot q} \tag{5.6}$$

The fragment shader code for calculating the acceleration looks like follows, where first the velocity and volume of the fluid element are retrieved, before the sum of all external forces is converted into the acceleration using the density of the liquid and the fluid volume:

```
vec3 acceleration = (invWorldMatrix * vec4(force, 0)).xyz / (density * volume);
```

Because we act in image space, the resulting 3D vector $\vec{a}$ here is first transformed into tangent space via $\vec{a'} = \left(\vec{T}, \vec{B}, \vec{N}\right)^T \cdot \vec{a}$ using the pre-calculated maps (see Figure 5.15 (ii)), and then projected into texture space by omitting the third coordinate: $\vec{a_p} = (a_x', a_y')^T$. As opposed to [112], instead of $\vec{a}$ the new velocity $\vec{v'}$ is then scaled by a gaussian-like damping function $\mathrm{r}(q) \in [0, 1]$, which allows to slow-down the velocity as well. Here, the less fluid remains on the surface, the stronger frictional forces are. Besides this, $\mathrm{r}(q)$ also allows modeling different frictional forces and viscosities (cp. Figure 5.13).

After having calculated the new velocity vectors, in the second sub-step the fluid volume is moved accordingly. Here, the transported volume is given by the displacement vector $\vec{d} = \Delta t \cdot \vec{v}$. As mentioned, fluid transport is achieved implicitly by considering the current fragment (grid cell $(i; j)$) and its eight neighboring fragments. Due to the implicit approach, a texel can only gather fluid from its direct neighbors. This is ensured by increasing the iteration steps for higher velocities.

The new amount of fluid is the sum of the gathered eight incoming fluid portions and the remaining fluid volume. Therefore nine texture look-ups are required with the constraint to not exceed the maximum distance of one grid element. The result is accumulated to the new fluid volume q' and again encoded as color for updating the surface texture map as shown in the following GLSL code fragment:

```
gl_FragColor = vec4(velocity.x, velocity.y, volume, 0);
```

5.3.3 Rendering of Droplets

The result of the simulation is now readily available in the surface map for visualizing the droplet flow in the final rendering pass. Thus, in this section, the visual appearance of fluids like tears and sweat will be handled, because it is solely located in image space and therefore can be implemented on the GPU.

Here, we have to deal with different levels of complexity depending on the type of fluid, the optical properties of the underlying surface, and the distance to the viewer. In the latter case, if the viewpoint is very close to the drops, we also have to take surface reflections into account. The most noticeable attribute of sweat drops is the refraction of light and development of strong highlights, which are common for fluids.

The treatment of reflections and refractions requires creating environment or cube maps in additional rendering passes, and thus can lead to poor performance for complex rendering situations. To get a refracted image of things behind the sweat drops, a first rendering pass of the background image is taken from camera view. Then the refraction vector is calculated with an index of refraction $\eta \approx 1.33$ for water following Snell's law. Fortunately, handling mirroring effects is only necessary for very close-up views, and handling complex refractions is only important for rendering transparent fluids and surfaces.

The opaque, green slime shown in Figure 5.13 is the easiest case for rendering. Nevertheless, on-the-fly gradient computation is required for all cases. Therefore, the blue channel of the surface map, which contains the fluid volume q, is treated as a height map. Analogous to the common algorithms for creating bump maps, the gradient ∇q or tangent space normal respectively, is obtained via the central differences method.

When having the normals, for powerful highlights a Fresnel term and simple Phong shading with a large specular component can be applied for nice, specular highlights. The surface map thereby additionally serves as specular map, whereas the amount of specularity, for example in case of a clammy forehead due to sweating, is automatically correctly updated. Because refraction is only noticeable in close-up views, if only the fluid but not the surface is transparent (as in the case of the tears), for a good quality–speed trade-off it can be approximated by just taking the diffuse color or texture of the underlying surface [155] when the viewpoint is a bit further away.

The examples showing water drops on a window glass (Figure 5.13, left, and Figure 5.19) also require to get a refracted image of things behind the drops. Therefore, another rendering pass of the scene behind the surface is taken from camera view. Similar to shadow mapping techniques, this image is then mapped to the transparent surface via projective texturing for being correctly aligned with the scene.

Figure 5.19: *Photo of real rain drops on a window pane (left) and two frames showing simulated rain drops on a window glass with view onto a foggy and rainy landscape (right).*

Following the techniques for simulating refractions described in [294], in the next step a perturbation of the texture coordinates is introduced for achieving the refractive look. Thereto, the gradient is scaled by a small value and added to the projected texture coordinates for displacing the texture lookup accordingly, as shown below. As opposed to calculating the real refraction vector, this method likewise is not physically-based, but it leads to convincing results and is very efficient.

```
vec3 projectiveBiased = (projCoord + s * normal).xyz / projCoord.w;
```

Tears can be handled similarly to sweat, with the difference that they look quite unrealistic when not running down the face (Figure 5.12, bottom). Since on graphics hardware that does not support SM 3.0, FBOs and floating point textures a simulation is computationally too intensive, once again we exploit 3D textures, but this time they are built up with either artistically created or pre-simulated normal and gloss maps, the latter encoded in the texture's alpha channel (Figure 5.8, right, on page 145). In a similar manner each layer refers to a frame of a "cry" animation. By changing the z-coordinate continuously from 0 to 1 a smooth animation just like in a cartoon is achieved. This method, but with red as base color, does also work well for flowing blood.

5.3.4 X3D Integration and Discussion

The proposed nodes are integrated into the InstantReality framework [135] that uses OpenSG [232] for rendering. For simplifying the usage of the described system, we have implemented a special *DropletFlowAppearance* X3D node. It inherits from *X3DSimulatedAppearance*, which likewise extends the X3D ISO standard [336] for handling all simulations that run in image space. The node interface is shown next. As mentioned, the simulation result is a texture that only contains the velocities and volumes. Hence, it is not suitable for being displayed without further evaluation, which requires deeper knowledge of simulation details. Thus, instead of using a *Texture* node, we have integrated the simulation system as part of a special *Appearance*, which hides the complexity and also allows designers etc. to use the simulation.

The *DropletFlowAppearance* can either be used instead of the X3D *Appearance* node or as an additional appearance node within a *MultiPassAppearance* [135]. Whereas in the

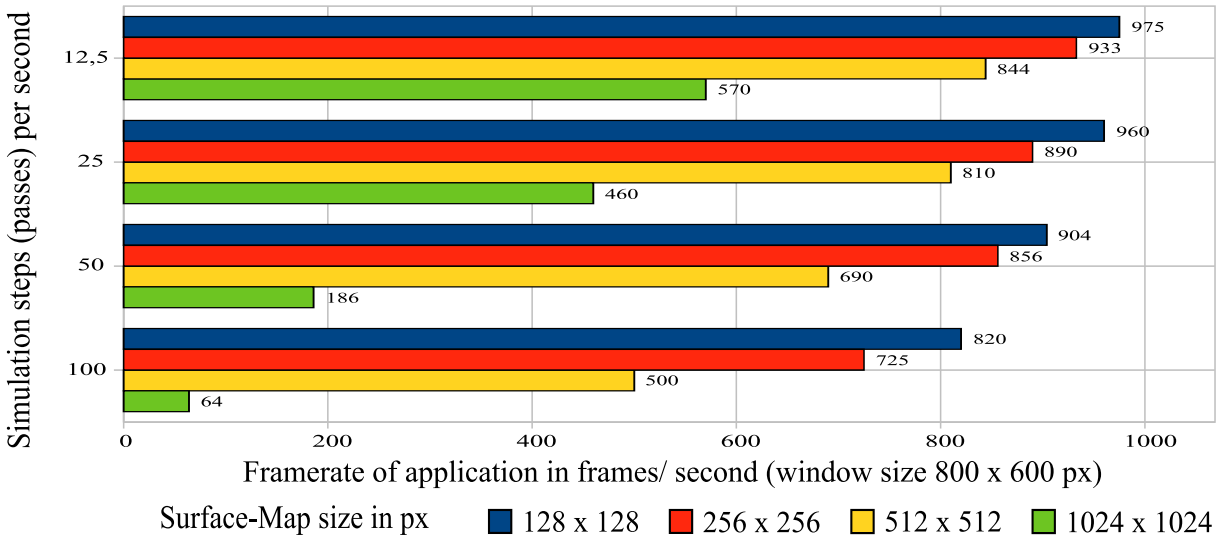

Figure 5.20: *Simulation performance for different surface map sizes on a GeForce 8600 GT.*

first case a texture can be provided and all shading is handled internally, in the second case already existing materials remain unchanged. Furthermore, because here the droplet flow is simply blended onto the existing *Shape*, multiple *DropletFlowAppearance* nodes with different parameters can be stacked for achieving more complex effects.

Most of the node's fields should be self-explanatory. If the 'scene' field is not set, the *Shape* node's geometry is used as surface. Different shading modes (e.g. "multiPass" or "singlePassPhong") can be chosen via the 'appearanceMode' field. For advanced users, there is also the possibility to directly use the surface map for writing own shaders, by providing a target texture and setting the 'appearanceMode' to "none". Via the 'sources' field drops can be placed by specifying uv-coordinates and sizes.

```
DropletFlowAppearance : X3DSimulatedAppearance {
  SFNode      [in,out] texture        NULL
  SFNode      [in,out] scene          NULL
  SFTime      [in,out] timeStep       0.1
  SFVec2f     [in,out] surfaceMapSize 256 256
  SFFloat     [in,out] perturbAngle   0.1
  SFFloat     [in,out] friction       0.5
  SFFloat     [in,out] fluidDensity   1.0
  SFFloat     [in,out] fluidViscosity 1.0
  SFFloat     [in,out] contactAngle   1.570778
  SFColorRGBA [in,out] fluidColor     0.1 0.2 0.9 1.0
  SFVec3f     [in,out] force          0.0 -1.0 0.0
  MFVec3f     [in,out] sources        []
  SFString    [in,out] appearanceMode "auto"
  SFBool      [in]     pause
}
```

To convey certain emotions not only the characters' emotions but also lighting and other atmospheric phenomena such as fog and rain should be considered. Therefore, our previ-

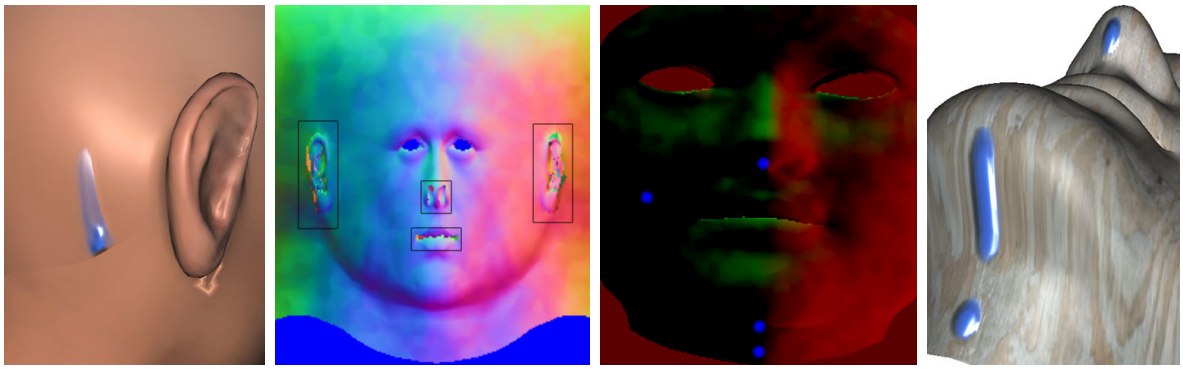

Figure 5.21: *Problems caused by texture mapping: (i) At texture seams droplets suddenly "leave" the surface. (ii) If uv-mapping is not one-to-one, a drop can appear multiple times. Distorted texture coordinates in surface map (iii) can cause rendering errors (iv).*

ously described method for real-time droplet flows on 3D surfaces can be utilized. Because at the X3D level nodes can serve many purposes depending on the application, our new "DropletFlowAppearance" node cannot only be used for simulating weeping or perspiration, but also for simulating rain drops on a window pane (e.g. in a flight or car simulator) or wet color running down a wall (e.g. in a virtual graffiti application). Figure 5.19 shows some frames of water drops running down a window glass (Figure 5.19), which requires getting a refracted image of things behind them, in this case a foggy landscape.

The measurements shown in Figure 5.20 were taken on a dual core PC with an NVidia GeForce 8600 GT graphics card. As can be seen, real-time framerates are achieved. But since FBOs and floating point textures must be supported, at least a Shader Model 3.0 capable GPU is required. Some results are shown in Figures 5.13, 5.18, and 5.19. The images to the left in Figure 5.18 show three frames of the droplet simulation applied to a terrain, being subject to gravity. Figure 5.19 shows a window pane with rain-drops running down that refracting the scene behind them, whereas the image to the left shows a real photograph for comparison and the images to the right are dumped frames from our simulation.

However, the proposed method also has drawbacks and limitations that are left for future work, most of them due to texture mapping problems as visualized in Figure 5.21. Thus, only objects with one geometry and continuous texture coordinates (as opposed to a texture atlas) can be used. In addition, a texel can only contain a certain amount of fluid. As soon as this limit is reached, the excess amount of fluid gets lost. Moreover, the transition from image to object space, i.e. fluids dripping off a surface, is currently not possible. A possible solution could be to use stream out to vertex and pixel buffers in combination with geometry shaders.

5.4 Conclusions

In this chapter we first described extrinsic factors of skin rendering, incorporating properties of human skin and aging as well as reflection and scattering in general, and discussed a possible X3D integration. For modeling behavioral aspects intrinsic factors were con-

sidered, too, including rendering and animation of emotions. Therefor, we've presented a parameterizable emotion model for representing skin changes for virtual characters in the context of dialog systems and an evaluation of the results. To define the emotion model, a classification of possible skin changes based on physiological and psychological knowledge was performed and verified in an experimental study, which has demonstrated that also considering vegetative functions is essential for a correct perception of certain emotions.

Integrating psycho-physiological reactions into multimodal dialog systems or computer games thus includes having the ability to control them just like voice and motor response, and also requires having appropriate building blocks on the graphics side like a real-time crying simulation and appropriate rendering techniques (cf. section 5.3). It was also shown that by combining standard facial expressions with complexions and similar effects such as sweating and crying, the corresponding emotion is easier perceivable and for intense emotions also more plausible and consistent. By incorporating techniques for visualizing emotions, and therefore providing consistency with facial expressions, an important step towards lifelike and believable virtual humans is done.

All shading methods presented in this chapter are running on modern graphics hardware at real-time frame rates, and the algorithms are kept very general and not specific to a certain character or application. Apart from the droplet flow simulation, which requires at least Shader Model 3.0, the other algorithms can already be implemented for GPUs only supporting SM 2.0. Moreover, the proposed techniques utilize the current X3D standard and, where necessary, extend it appropriately. They are suitable for scenarios such as a dialog system, where only a few virtual characters have to be rendered in a realistic way, because they are heavily based on multi-pass techniques. Example applications are games, e-learning environments with virtual guides, or cultural heritage projects.

The emotions described in Table 5.1 are very strong and normally not used in e.g. tutoring systems and other industrial applications. However, simulating those emotions or facets of them in combination with mimics and body poses is not only important to convey feelings, but also helps making virtual characters more responsive and plausible. As an example, it is quite unrealistic, if in a computer game an injured character neither cries nor turns pale or cold sweat appears on his forehead.

Here, the expression of feelings is represented through avatars, and also exaggerated emotions can be envisioned. Basic and strong emotion visualization is furthermore important in edu- or infotainment applications, where children and youngsters should learn to reflect strong facial emotion. The goal here is to increase their ability to emphasize and to learn handling conflicts without violence, as today e.g. is done in infant schools.

At the moment the introduced framework for creating skin changes only applies to the face of a virtual character. An extension to the whole body, e.g. to simulate goosebumps via bump maps, would be useful. In combination with the corresponding, here fearful, posture, this permits more realistic emotion visualizations even for close-up views. For future work, also an in-depth evaluation of the results in collaboration with psychologists and physicians would be very useful. Furthermore, it should be investigated, if the discussed phenomena vary between different races and/ or cultural environments.

6 Camera and Lighting

This chapter first deals with enhancing camera models and camera control in virtual environments [150, 149, 20, 153, 160]. After that, it is discussed how shadows and other modern rendering techniques can be integrated into X3D [151, 156, 23, 88]. Finally, the presented techniques are utilized to almost seamlessly integrate virtual objects into real scenes in the context of Mixed Reality applications [151, 89, 88, 262], where virtual characters can represent a natural interface in contrast to standard WIMP interaction.

6.1 Introduction

Current frameworks that employ virtual agents often rely on non-standardized pipelines and lack functionality to describe lighting, camera staging or character emotions in a descriptive and easy to use manner. There are currently no standards and components for behavior modeling and rapid authoring of interactive content. Even though demand for such a system is high, ranging from edutainment to pre-visualization in the movie industry, few such systems exist. In this regard, we make use of X3D [336] for scene description and runtime environment, since representing all assets as X3D means that they are easily distributable and sharable to others. We also look into techniques that can additionally be used to convey certain emotions within virtual environments, which will lead to better immersion as concepts similar to those in the world of film are introduced and made available to interactive X3D applications.

Whereas skin color change and droplet flow to simulate effects like weeping, including a parameterizable model to classify emotions, were discussed in the previous chapter, here first rendering methods for lighting as well as a cinematographic camera approach to enhance these effects are described. Thus, e.g. [56] pointed out that even the choice of lights, shadows, camera and lens filters can influence a user's perception of emotions. Hence, the environment can be used for communication, too, in that factors like lighting and camera control can be utilized to define, clarify, or emphasize a character's personality, role and interpersonal relations. Furthermore, as already was outlined in section 3.3, simulating realistic lighting conditions is also essential for Mixed Reality scenarios, where the avatar and other virtual objects are inserted into a real scene.

Hence, three major elements need to be re-examined from a cinematic point of view [20, 160]. Characters, as the first element, have been discussed in the previous chapters. While posture and mimics have been subject of extensive research in various areas like CG and AI, here we are focusing on dynamic skin changes in order to be able to express strong emotions, too. To achieve plausibility and consistency between different modalities not only the integration of elements like high-quality rendering and animation generators

is required (including models for automatically generating realistic communicative behaviors), but also the ability to control posture, voice, and mimics from a higher level, to allow going all the way from communication models to rendering.

Second, a cinematographic camera approach is often necessary to focus on important elements [150, 153] and to illustrate certain subtle effects like crying that may be only visible in close-ups [149]. Third, we make use of Mixed Reality (MR) techniques to obtain more realistic lighting results for both, Virtual and Augmented Reality applications [160]. Hence, after discussing suitable illumination techniques in general, which are also an important factor for expressing moods [56] from a decoding perspective (on linguistic grounds), in section 6.4 we will especially focus on utilizing these techniques in the context of Mixed Reality applications.

6.2 Virtual Cameras

Creating and setting the right parameters for the virtual camera is crucial for any content creation process. However, this is not easy since most current camera models, including the X3D *Viewpoint*, use a 3D position and orientation in 3D space to define the final visualized image. People use authoring tools or simple interactive navigation methods (e.g. "lookAt" or "showAll") to ease the process but at the end they still move a 6D (translation and rotation) camera beacon to get the final image. We thus propose a new X3D camera model, the *CinematographicViewpoint* node, which does not force the content creator to move the camera but allows the scene author to directly define what objects he/ she would like to see on the screen [150].

We borrow established techniques from the film area (e.g. the rule of thirds and line of action explained in section 2.3) that allow defining objects and object-relations, which the camera model will use to automatically calculate the final transformation in space. The new camera model includes additionally a model for global visual effects (e.g. motion blur and depth-of-field), which allows incorporating classical film effects to real-time scenes. Both approaches combined allow content creators building visual results and camera movements that are closer to traditional filming much easier. Our approach was proposed in [150, 149, 20, 160], and also supports automatic camera movements that are bound to interactive content, which has not been possible before.

6.2.1 A Declarative Approach

Film and games industries are converging more and more – not only in the way that their creative content is perceived by consumers but also because of various IP rights. Hence, the ANSWER project [6] is a new approach to the creative process of film and game production. Its main goal is to produce a symbolic language, the *Director Notation* [349] (that currently consists of two parts: Acting and Camerawork Notation; cf. Figure 6.1, right), for describing the creation of 3D content especially in the context of film making as well as for the authoring of cut-scenes in game design or machinima. This idea is similar in workflow and required tool sets to setting up e.g. an infotainment system.

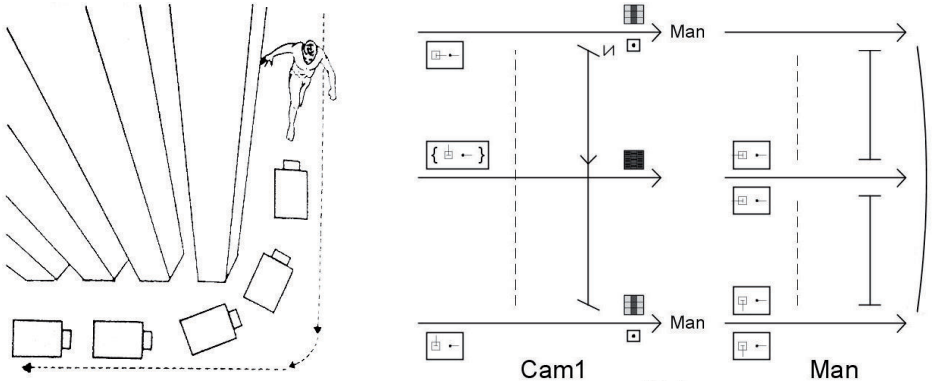

Figure 6.1: *Left: A sketch of a short shot taken from a director's storyboard. Right: The same scene in the formal DirectorNotation with framing symbols as proposed by [349].*

So today, the process of film production is still lengthy, time consuming, and expensive, and film directors currently rely on paper based storyboards (see Figure 6.1, left) to express their creative intent on how a scene could be shot by using rough sketches on a piece of paper and their imagination. There is still no formal way to describe this intent. To alleviate this, a new notation system for describing cinematic content is provided, offering a bridge between film and digital media production and animation for game design. An ontology tries to model the film domain in a way that is coherent with the director's intent during film production, and the concrete specification of such an ontology is presented to the user in the form of Director Notation.

The ANSWER framework uses this underlying semantic model and provides a set of interconnected components that aid a director in the process of film production from the planning stage to post-production [20, 160]. Intuitive interfaces are used for authoring and the underlying knowledge model is populated using semantic web technologies over which reasoning is applied. This transforms the user input into animated pre-visualizations that enable a director to experience and understand certain film making decisions before production begins. Albeit at a first glance this topic does not seem to be directly related to multimodal dialog systems, for the presentation module the same requirements apply, because the major challenges concerning real-time visualization are not only the generation and control of plausible character animations from a higher level, but also the consideration of camera movements, lighting, and other atmospheric effects.

Directors and other creative people tend to think in images and complete scenes rather than the details of how they are achieved. An illustrative example on how a director works and thinks is the famous Holodeck. In one *Star Trek* episode Commander Riker asks for a pub scene, Earth, Chicago, early 20th century, a piano to the right, tables in the middle, and some people sitting around. In this context the Holodeck can be considered as a really advanced content creation tool. Although we are still far away from this dream, what we can learn from this example is the need for descriptive interfaces as well as for more target oriented virtual cameras with functionalities beyond a simple "lookAt" or "showAll" for easing content creation.

Hence, in this section we describe our declarative approach to camera placement and its embedding into the ISO standard X3D via the *CinematographicViewpoint* camera

node. Because X3D already allows defining scene description and runtime behavior of a 3D scene-graph on a descriptive level, this approach seems to be a natural extension to X3D [150, 341, 160]. It is motivated by the fact that creating and setting appropriate parameters for the virtual camera is crucial for any content creation process. However, this is not easy since most camera models, including the X3D *Viewpoint*, do not consider what shall be framed, but instead use a 3D position and orientation.

In Virtual Reality, computer games, and 3D graphics in general, navigation means to interactively change camera parameters by directly modifying 3D coordinates, and to thereby define which details of a scene we want to see. This approach is unlike in filming, where the director exactly specifies what shall be visible on the screen. Because this is much more natural for users and also an important use case for storyboarding and early pre-visualization [6], we thus follow the converse approach and present a new camera model that allows framing objects just like in film.

Moreover, our camera model additionally provides a model for including special effects, which are quite important for the perception of a film and usually depend on the camera system used, like motion blur and depth-of-field, in order to allow incorporating classical film effects into real-time scenes easily. Furthermore, effects like blur are also useful in the context of AR/ MR, e.g. in order to prevent virtual objects from being too sharply silhouetted against the camera image.

Cinematographic principles and terminologies were already described in section 2.3.2 on page 59. For example, when using a close-up shot even subtle effects like tears can be framed, as shown in Figure 6.2 (v). However, as was also mentioned, e.g. an over-the-shoulder shot already requires semantic knowledge, like which part of the character's mesh denotes the shoulder, and hence is not handled on this level, since knowledge engineering is out-of-scope here. Nevertheless, possible properties of the camera system need to be exposed to higher layers in a declarative way [149].

Likewise X3D only provides lightweight components for storage, retrieval and playback of real-time 3D graphics content embedded in applications [336] – but not the applications itself. Thus, our proposed camera node (which will be explained in more detail in the following section and whose design and implementation is mainly based on the concepts presented in [119]) provides all necessary low-level functionalities for enabling application developers or high-level modules to define how the final images shall look like.

6.2.2 The CinematographicViewpoint Node

The proposed *CinematographicViewpoint* [150] is a specialized perspective camera node for handling cinematographic requests like the rule of thirds and is integrated into the X3D-based Instant Reality framework [135]. The new viewpoint inherits from *X3DViewpointNode* node, and its interface is shown next. Many fields, like 'set_bind' (sending TRUE makes this node active and FALSE makes it inactive) or the 'fieldOfView' field (preferred minimum viewing angle from the viewpoint in radians – a small value roughly corresponds to a telephoto lens, and a large one to a wide-angle lens) are the same as in the standard X3D *Viewpoint*, which is shortly discussed in section 6.2.4.

Figure 6.2: *The rule of thirds – respectively the two cases depicted in Figure 2.9 – applied to different models. From left to right: (i) a sphere and (ii) a sitting character in the middle of the screen; (iii) a full shot and (iv) a close-up shot of a character in the top left of the screen (for calculating the camera positions, different bounding volumes, shown in red, are used); (v) an extreme close-up for emphasizing the tears.*

```
CinematographicViewpoint : X3DViewpointNode {
  [...]
  SFFloat    [in,out] fieldOfView    0.785398
  MFNode     [in,out] objectsFull    []
  MFNode     [in,out] objectsCloseUp []
  SFVec3f    [in,out] facingDir      0 0 1
  SFVec3f    [in,out] upVector       0 1 0
  MFVec2f    [in,out] minScreenPos   []
  MFVec2f    [in,out] maxScreenPos   []
  SFString   [in,out] shotSize       "auto"
  SFFloat    [in,out] shotAngle      0
  SFFloat    [in,out] shotPitch      0
  SFFloat    [in,out] shotRoll       0
  SFString   [in,out] follow         "none"
  MFNode     [in,out] effects        []
}
```

The node's fields are explained in the following. The 'objectsFull' field takes a reference to the whole object and is used to determine the object that is taken for calculating the full view on the objects of interest, based on the bounding box. Likewise, 'objectsCloseUp' takes a body part such as the head and is used to determine the close up view (currently maximal two objects may be defined here). Both fields in combination with 'facingDir' are also used for calculating the line of action.

The latter field is not only used to adjust the line of action for one character (usually the local facing direction is along the z-axis, but modeling tools may have exported differently), but also to determine, which side of the line of interest given by two actors shall be used (cp. Figure 2.8). Therefore, the camera has to respect the constraints of not crossing the line of action and thus maintaining continuity. By interpolating smoothly between succeeding camera poses, this offers a fluent and continuous camera movement comparable to cinematic camera movements.

The SFString field 'follow' determines the camera node's behavior concerning object tracking. If the field value is set to "all", the camera continuously follows the target object/ actor, which is shown in Figure 6.3. This can be further restricted by only evaluating the

Figure 6.3: *Frames showing a walking character with different shot sizes. Whereas in the left-most images the camera retains a close-up view upon the face throughout the motion, on the right side the camera always faces the full model to be in the middle of the screen, as symbolized in Figure 2.9 (left) on page 61.*

aiming position or orientation (via "onlyPos" or "onlyOrient" respectively). By using the value "none", the camera only calculates the appropriate position and orientation when it is activated (i.e. on bind time).

The field value "onlyOrient" is especially useful in combination with a camera movement called camera track [341]. Here, the camera is moved along a certain unconstrained path, whilst always facing the target objects, which thereby results in continuous changes of the camera orientation. When the object also moves at the same time, jitter can occur due to the permanent orientation adjustments. This can be resolved by internally smoothing the resulting orientation analogous to an X3D *Chaser* node [336] by simply averaging the last n camera rotations [160].

The two fields 'minScreenPos' and 'maxScreenPos' allow setting the minimum and maximum bounding box position of an object in normalized screen coordinates (for enabling to specify positions in screen-space according to the rule of thirds). The two cases depicted in Figures 2.9 and 6.2 (left and middle) for instance can be specified by setting 'minScreenPos' and 'maxScreenPos' to the ranges $[\frac{1}{3}, \frac{1}{3}] \times [\frac{2}{3}, \frac{2}{3}]$ and $[0, \frac{1}{3}] \times [\frac{1}{3}, 1]$ respectively.

As can be seen, the application developer is not necessarily bound to some default locations but is free to specify any position on the screen. Because specifying screen locations this way might by cumbersome sometimes and not very intuitive, in addition the final positions can be modified by using the 'shotSize' field, e.g. as shown in Figure 6.2 (v) for emphasizing the region between the actor's mouth and eyes.

Because bounding spheres are computationally most efficient here, we use two of them for calculation as proposed in [119]: one for close-up and one for full view, by considering the projected area of the sphere on the screen for obtaining the camera distance d. With the focal length f, the desired screen location $c_s = (x, y)$, and bounding radius r_w (in world coordinates) as well as r_s (in screen coordinates), the distance between camera and object is computed via $d = \frac{r_w}{r_s}\sqrt{x^2 + y^2 + f^2}$.

The camera then is initially placed at position $e = c_w + d \cdot shot_{size} \cdot line_{act}$ (with c_w being the object's center) and oriented accordingly by aligning it with the line of action (e.g. the character's facing direction) and finally "adding" the angle offsets, thereby obtaining a much more intuitive usage. The angle offset from the line of action can be modified with

the help of the 'shotAngle' field. Analogous, the angle offset to the horizontal plane is modified by means of the 'shotPitch' field, whereas 'shotRoll', a rotation of the up-vector along the z-axis, is not very commonly used for cameras.

Via the slot 'shotSize' the shot sizes can be defined for the first actor. As proposed in [119], possible values are "auto", "extremeLong", "long", "full", "mediumFull", "medium", "mediumClose", "close", "wideCloseup", "closeup", "mediumCloseup", and "extremeCloseup". If for instance 'shotSize' is "full", the object from the 'objectsFull' field is taken (as shown in Figures 6.2 (ii)/(iii) and 6.3, right), and starting with "close" up to "extremeCloseup", the object from the 'objectsCloseUp' field is taken for determining the object size (see Figures 6.2 (iv)/(v) and 6.3, left).

6.2.3 Camera Extensions for MR

For MR applications, the X3D Viewpoint node likewise is not sufficient. The Viewpoint node implements the classic pinhole camera model. Real video cameras do not follow this simple model. Because we integrate virtual objects into video images, we need to use a camera model that simulates the real camera as closely as possible. Therefore we present some extensions to the standard X3D *Viewpoint* node for fulfilling the requirements of Augmented and Mixed Reality applications as well as a new node that allows directly specifying the projection and modelview matrices [151].

The Viewpoint node defines a perspective view of the scene, where all projectors are coalesced on a single position in space. The "position" and "orientation" fields define the camera transformation and the "fieldOfView" field specifies a preferred minimum viewing angle from this viewpoint. This abstraction fulfills the needs of common desktop-based applications. Real-world cameras are far more complex and therefore AR applications require comprehensive viewing models, which include the inner and outer orientation of the device [118]. Accordingly, we provide an extension to the X3D *Viewpoint* node and additionally a new *Viewfrustum* node, which give the application developer more freedom.

```
Viewpoint : X3DViewpointNode {
  [...]
  SFString [in,out] fovMode        VERTICAL
  SFVec2f  [in,out] principalPoint 0 0
  SFFloat  [in,out] aspect         1.0
}
```

The new fields provide a more general camera model than the standard *Viewpoint*. The "principalPoint" field defines the relative position of the principal point. If the principal point is not equal to zero, the viewing frustum parameters (left, right, top, bottom) are simply shifted in the camera's image plane. A value of $x = 2$ means the left value is equal to the default right value. A value of $x = -2$ means the right value is equal to default. If the principal point is not equal to zero, the "fieldOfView" value is not equal to the real field of view of the camera, otherwise it complies with the default settings.

To extend this idea, the "fovMode" defines whether the field of view is measured vertically, horizontally or in the smaller direction, which is important for correctly parameterizing

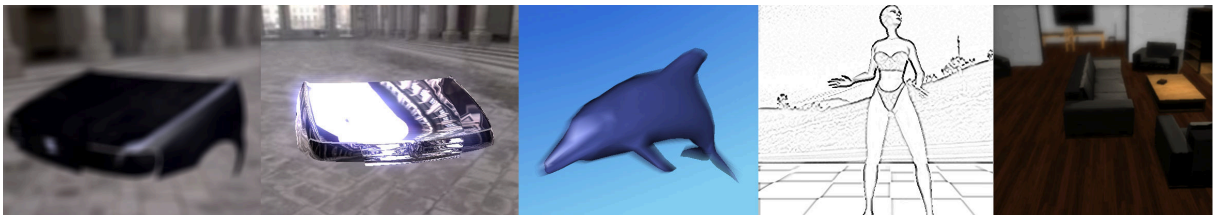

Figure 6.4: *Some post-processing effects that can be enabled via the new 'effects' field: (i) blur, (ii) glow, (iii) motion blur, (iv) sketch, (v) depth-of-field.*

the aforementioned cinematographic camera. The field "aspect" defines the aspect ratio for the viewing angle defined by the "fieldOfView" range. This setting is independent of the current aspect ratio of the window, but reflects the aspect ratio of the actual capturing device. This extension allows us to model cameras with a non-quadratic pixel format, i.e. it defines (width / height) of a pixel.

In addition to the *Viewpoint* extension we include a new camera node named *Viewfrustum*. This node has the two input/output fields "modelview" and "projection" of type SFMatrix4f. With the *Viewfrustum* node we are able to define a camera position and projection utilizing a standard projection/ modelview matrix pair [174].

This encoding is quite common in today's online visualization and tracking systems and eases the integration of existing systems as outlined in section 6.4.6.1. Likewise we also propose a generic *MatrixTransform* node with an input/output field "matrix" of type SFMatrix4f, which can be used as parent node of the *Viewpoint* or a tracked object.

6.2.4 Special Visual Effects

In this section we present some nodes for realizing special visual effects [150] that are suitable to either artistically modify the visual output or mimic real-life camera properties and which likewise can enhance the rendering quality or create certain moods.

As stated in [56, 45], not only framing but also the correct choice of lenses, filters and other effects applied during post-production are crucial for the final perception. Because this is impossible to do with current X3D, the camera extensions proposed in [341, 150, 160] additionally include a model for global screen-space effects such as motion blur and depth-of-field to incorporate classical film effects. Similar in spirit, for SVG (a format for describing two-dimensional vector graphics in XML) there exists a draft specification for SVG Filters [324], which can be used to filter the input e.g. to become black-and-white or to produce shadows.

Moreover, especially in combination with Mixed Reality applications (see section 6.3) it can be essential to appropriately choose a suitable filter. If for instance one has a blurry camera image, it looks very unrealistic if the augmented 3d scene is pin sharp. Thus, by applying some sort of blur filter and/ or grain to the virtual objects, the impression of seamless integration can be enhanced a lot.

Hence, our proposed camera additionally includes a model for global screen-space effects such as motion blur and depth-of-field, which besides this also allows incorporating classi-

Figure 6.5: *Same scenario as in Figure 6.9 (with 4 derived lights) but now with a character. From left to right: no depth-of-field, character nearby, focalDepth=600; character nearby, focalDepth=12; character further away.*

cal film effects to real-time scenes quite easily. Figure 6.4 shows examples of some special effects computed in a post-processing pass, which internally is achieved by first rendering the scene into a backbuffer and then filtering the resulting images by means of fragment shader programs [151, 114, 264].

Because the mentioned visual effects usually are specific to a certain type of camera or lens system, we propose to extend the standard X3D *Viewpoint* nodes with an additional MFNode field 'effects' (as shown above) for specifying the desired global effects like depth-of-field and motion blur for the post-processing step. Therefore, we introduce a new abstract base node, the *X3DVisualEffects* node whose interface is shown next, for handling a special type of effect. By defining more than one effect at a time additional effects can be achieved, depending on their order as given in the 'effects' field. The concrete realizations of this node (e.g. *BlurFX*, *GlowFX*, *MotionBlurFX* and *SketchFX* as shown in Figure 6.4), additionally have fields for parameterizing the effects appropriately.

```
X3DViewpointNode : X3DBindableNode {
  SFBool     [in]     set_bind
  [...]
  MFNode     [in,out] effects  []
}
```

Alternatively, this can also be modeled as another type of *X3DBindableNode* for decoupling camera and effects. Additionally, we also have to think about montage and cuts. This can partly be achieved by extending the current mechanism of binding a viewpoint with new features for fading or brighten up the scene. Likewise, when animating from one camera to the next, also the corresponding effects have to be appropriately switched, e.g. by interpolating the focal depth in case of depth-of-field.

The cinematographic camera approach and subtle mimic effects and the like are complementary, as the latter are only visible in close-ups. The more realistic rendering appears, the more subtle differences in camera height, angle of view and in the position of actors can make a difference to the meaning of a scene. To avoid this problem, especially when dealing with low-cost assets, non-photorealistic rendering techniques can be used, e.g. by utilizing the *SketchFX* node for doing sketch-like shading.

Moreover, non-photorealistic rendering can be an important feature not only to avoid the well-known Uncanny Valley effect (especially in combination with perceptually unsatisfying character motions [207]), but also to be able to focus on the essential aspects of a scene that often are better visualized on a more abstract level, because too much quality can distract from the main ideas as the type of rendering already implicitly directs the user's thoughts. This has the additional advantage that a scenario can be rendered and animated rather simplistically to fulfill real-time constraints and to keep time and effort concerning asset creation at a minimum.

```
X3DVisualEffects : X3DNode {
  SFBool  [in,out]  enabled TRUE
}

SketchFX : X3DVisualEffects {
  SFBool  [in,out] enabled   TRUE
  SFInt32 [in,out] thickness 1
}
```

Most of our proposed node interfaces extend the work of [150, 341] and are suited for enhancing rendering quality by handling screen-space post-processing effects [160]. Basically we distinguish between two types. For one thing, there are nodes for enhancing the general quality by exposing appropriate functionalities like ambient occlusion, and for another thing we have pure effects like the aforementioned sketch rendering. But in general, it is a smooth transition as e.g. the *HDRRenderingFX* not only provides blooming effects, but thereby also enhances rendering quality. All FX nodes have an SFBool field 'enabled', which, like the 'metadata' fields, are omitted in the following descriptions for better readability.

```
DepthOfFieldFX : X3DVisualEffects {
  SFFloat [in,out] focalDepth 10.0
  SFFloat [in,out] blurCutoff 0.7
}

HDRRenderingFX : X3DVisualEffects {
  SFFloat [in,out] exposure            1.64
  SFFloat [in,out] brightnessThreshold 1.0
}

ScreenSpaceAmbientOcclusionFX : X3DVisualEffects {
  SFFloat [in,out] scale       0.001
  SFFloat [in,out] attenuation 0.001
}
```

Depth-of-field sets the range of distances where all objects appear with acceptable sharpness. The proposed *DepthOfFieldFX* node, whose implementation is based on [277], allows enabling depth-of-field effects as shown in Figure 6.5. After rendering the scene, the resulting image is filtered with a variable-sized filter kernel to simulate the circle of

Figure 6.6: *Utilizing the BlurFX effects node for an old-fashioned look of a chapel scene. From left to right: standard rendering, only blur with grain, black-and-white rendering, all effects combined (black-and-white, blur with kernel size 3, grain).*

confusion, which is controlled by a depth-dependent blurriness factor. Because this factor is based on the relative distance to the focus plane, the 'focalDepth' field determines the distance at which the scene is in focus, and the 'blurCutoff' determines the normalized scene depth, where the blur maximum is reached. As can be seen in Figure 6.5, depth-of-field for instance can be useful as an additional possibility to focus attention by blurring less important things.

The *HDRRenderingFX* node enables HDR rendering including glow and tone mapping, which is especially useful in combination with the new lighting nodes explained in section 6.3. Here, the exposure is controlled by the correspondent input/output field, whereas all intensity values above 'brightnessThreshold' will be blurred for achieving glow effects.

Moreover, for improving quality beyond standard shadow mapping, via the *ScreenSpaceAmbientOcclusionFX* node screen-space ambient occlusion (SSAO) can be enabled. It coarsely approximates the occlusion term and was first presented by [220]. In contrast to classic ambient occlusion [245] it is independent of the scene complexity by mainly relying on the depth buffer. The node can be further parameterized via 'scale', which defines the neighborhood region in normalized screen-space used for occlusion calculation, and 'attenuation' for defining the depth attenuation.

```
MotionBlurFX : X3DVisualEffects {
  SFString [in,out] type      "auto"
  SFFloat  [in,out] strength  0.02
}

BlurFX : X3DVisualEffects {
  SFBool   [in,out] enabled       TRUE
  SFString [in,out] kernelType    "auto"
  SFInt32  [in,out] kernelSize    5
  SFBool   [in,out] grain         FALSE
  SFBool   [in,out] blackAndWhite FALSE
}
```

Another effect inherent to a certain type of camera is motion blur [264], which can be enabled via the *MotionBlurFX*. Whereas 'type' defines the type of motion blur (such as camera motion blur that only takes camera moves into account), 'strength' sets the blur strength in screen-space for scaling the sampled velocity vectors.

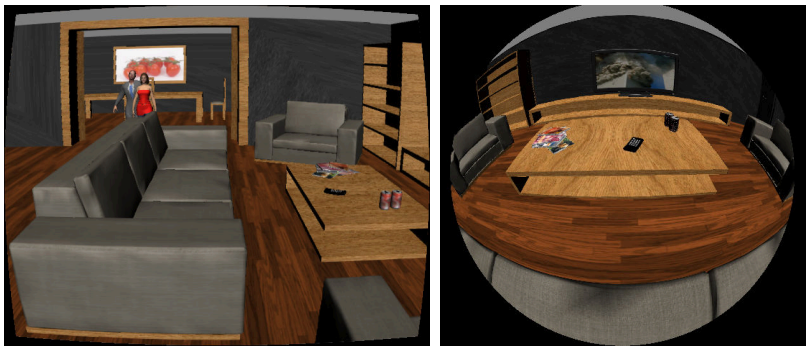

Figure 6.7: *Distorted view onto a living room scene (left) compared to strong distortion by means of a simulated fisheye lens (right) in order to allow for real-world lens types, too.*

The *BlurFX* node applies a blur filter to the final image and allows setting the type and kernel size of the filter to be used. In addition it has two other fields 'grain', to simulate old-style camera noise, and 'blackAndWhite', to enable grayscale rendering like in old black-and-white TVs. As can be seen in Figure 6.6, both effects can easily be combined to achieve an old-fashioned look for the final rendering. Similar techniques can be found in recent computer games like Left 4 Dead 2 to mimic the old B-movie style.

Additionally, we propose a new kind of effect for simulating special lens types that can distort the camera image, like the wide-angle lens shown in Figure 6.7, which exhibits a strong radial distortion. The latter can be calculated by means of a polynomial in the post-processing step, contrariwise to usual distortion correction as known from Computer Vision [118]. Therefore, for an undistorted point x we approximate the radial distortion by $x' = x(1 + k_1 r^2 + k_2 r^4)$, where r is the distance from the image center and k_1 and k_2 are measured distortion parameters.

The implementation of the fisheye effect shown in Figure 6.7 (right) is based on dynamic cube mapping, since we need to cover a 360° view. The cube map is regenerated every frame and then applied to a window-sized view-aligned quad within the *DistortionFX* node. This quad also makes use of a shader that evaluates the polynomial defining the fisheye lens for every fragment. Thereto, an appropriate lookup vector for indexing into the texture is calculated. To simulate the circular view of the lens, every fragment outside the radius of the simulated lens is set to black.

6.3 Set and Lighting

6.3.1 Environment and Stages

Having realistic lighting conditions is important to achieve the desired impressions [56], but in general requires global illumination, which usually is slow and expensive to compute. Instead, we take some concepts from Mixed Reality, where simulating real lighting conditions is essential for integrating virtual objects seamlessly into real scenes. One of the most important factors here is catching the real lighting situation for relighting the virtual objects. Therefore, different techniques can be used such as irradiance mapping

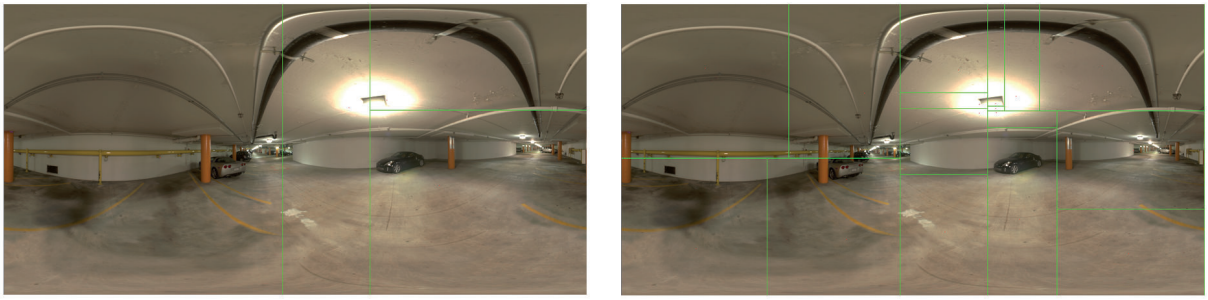

Figure 6.8: *Subdivision of a lat-long environment map into regions with same energy via the median-cut algorithm.*

[179] or the extraction of light sources from real footage [94, 209].

Using the usually spherical image of the surrounding environment not only for lighting (see sections 6.3.2 and 6.4.4) but also as the background image furthermore is a well-known method to improve the realistic look of a scene without an increase in geometric complexity. We therefore present a new type of X3D background node, the *SkydomeBackground*, which derives from the abstract *X3DBackgroundNode* [88].

The background texture itself is rendered on the inside of a sphere surrounding the entire scene. Because we need support for all types of environment maps, especially the latitude-longitude maps as explained next, this node has a field 'mapMode' for specifying the type of env-map used, and additionally it has an *Appearance* child node, which allows changing the final appearance or modification of the texture access within a shader program, as well as the implementation of a tone mapping algorithm if a high dynamic range image is used. In the following an example in VRML encoding describing the usage of the corresponding X3D node interface is shown.

```
SkydomeBackground {
  appearance Appearance {
    texture DEF irradianceMap ImageTexture {
      url "latlong.hdr"
    }
    shaders ComposedShader {...}
  }
}
```

6.3.2 Light Source Extraction

In addition, we propose a special X3D lighting node, the *EnvironmentLight*, in order to fit the lighting conditions of the environment map [160]. This node not only eases setting up more realistic lighting conditions as an important factor for expressing moods [56], but it is also rather useful in the context of Mixed Reality applications, where lighting reconstruction is still an open issue. Hence, a real-time approach to extract lights from HDR sphere maps for instance was presented in [303] and [190].

Accounting for complex lighting situations as given by the environment map comes with high computational costs. Instead of employing image-based methods, which don't work well with our aforementioned subsurface scattering approximation, we use the median-cut algorithm (see Figure 6.8) as originally proposed by [60, 209] for extracting the light sources, including their color and intensity (compare Figure 6.9), from the given environment maps. This has the additional advantage that afterwards standard shadow mapping techniques can be applied.

The basic idea of the algorithm is to recursively subdivide the image into 2^n regions with the same light energy (marked in green in Figure 6.8). Then for every region a light source is placed at the energetic barycenter. To be able to extract the light sources in real-time (which is only necessary when using video data as input) Summed Area Tables, which for a given entry contain the sum of all previous entries, are used for calculating the summed light energy of a region. Finally, for every region the 2D pixel coordinates, which in fact are spherical coordinates due to the lat-long format, of the energetic barycenters are transformed into 3D cartesian direction vectors for obtaining the light direction.

Hence, the *EnvironmentLight* node creates directional lights based on a given environment texture, which in our implementation currently must be provided in latitude-longitude format. Typically, this should be the same texture as used in the previously described *SkydomeBackground* node. For illustration, in Figure 6.9 a simple setup with diffusely gray geometric primitives and a HDR environment map is shown (which moreover affords having support for HDR image formats like Radiance and OpenEXR in X3D). For comparison, in Figure 6.5 the same setup is used, but instead of the primitives, a virtual character is used and depth-of-field is enabled.

Our proposed light node derives from the abstract *X3DLightNode*, which is the base type of all X3D light sources. The node interface is shown below. The 'envTexture' field holds the environment texture that is used for extracting the light sources, and the field 'numLights' defines the maximum number of lights that are generated.

```
EnvironmentLight : X3DLightNode {
  [...]
  SFBool  [in,out]  on               TRUE
  SFColor [in,out]  color            1 1 1
  SFFloat [in,out]  intensity        1
  SFFloat [in,out]  shadowIntensity  0
  SFInt32 [in,out]  numLights        2
  SFNode  [in,out]  envTexture       NULL
}
```

The 'shadowIntensity' field determines the intensity of the shadows, the generated lights will throw, where 0 means no shadows at all and 1 is full shadow intensity. This field likewise extends standard X3D concepts and was implemented following our approach first presented in [151] – see next section 6.3.3 for an in-depth discussion on shadowing techniques in X3D. The other fields, like 'color', 'intensity', and 'on', are already defined in the light node's base class.

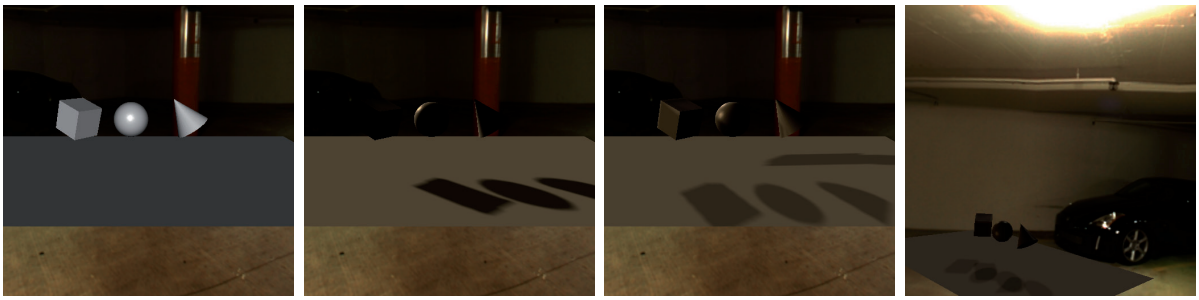

Figure 6.9: *From left to right: standard headlight; EnvironmentLight with numLights=1 (note the change in color); same with 2 extracted lights; EnvironmentLight with numLights=4 (from another view showing the real lights).*

6.3.3 Shadows

The current X3D specification does not include any kind of shadow for dynamic scenes. There are some proposals for shadow extensions, but most are limited in the types of lights or methods supported. Our extension [151] has two major advantages. First of all it works with almost every type of scene and is very intuitive. It also works with *Directional-*, *Spot-* and *PointLight*s and is independent from application defined shader programs. We do not introduce new special shadow nodes but extend the existing light nodes with additional fields. Therefore the light-node regulates illumination and shadows simultaneously, just like in the real world. Second, our parameter and abstraction level allows the support and implementation of different methods, e.g. shadow map or shadow volumes, to fulfill different requirements.

```
X3DLightNode : X3DChildNode {
  [...]
  SFFloat [in,out] shadowIntensity 0
  SFFloat [in,out] resolution      0.5
  SFFloat [in,out] mode            "auto"
}
```

The “shadowIntensity” field per light controls the shadow-casts for this light-node. The “shadowIntensity” (in $[0, 1]$) defines the intensity of the shadows. If the field value is equal to 0, the shadow is off, otherwise it defines the shadow opacity. The “resolution” is a mode-dependent scale factor, which determines the trade-off between speed and quality: 1 equals the best and 0 the fastest results for the given mode. In case of shadow mapping it defines the sampling range of the filter kernel.

The shadows are then automatically applied to all corresponding scene objects. The user does not need to specify specific occluders and occludees. But since we are in a virtual world, we can exclude objects from throwing shadows. This is quite useful for performance optimizations and can be achieved via the “shadowExcludeObjects” field of the new *Environment* node. This node is a bindable, which holds rendering states and settings like global shadow parameters. The most important shadow parameters are “globalShadowIntensity” and “shadowMode”. The “globalShadowIntensity” allows to force

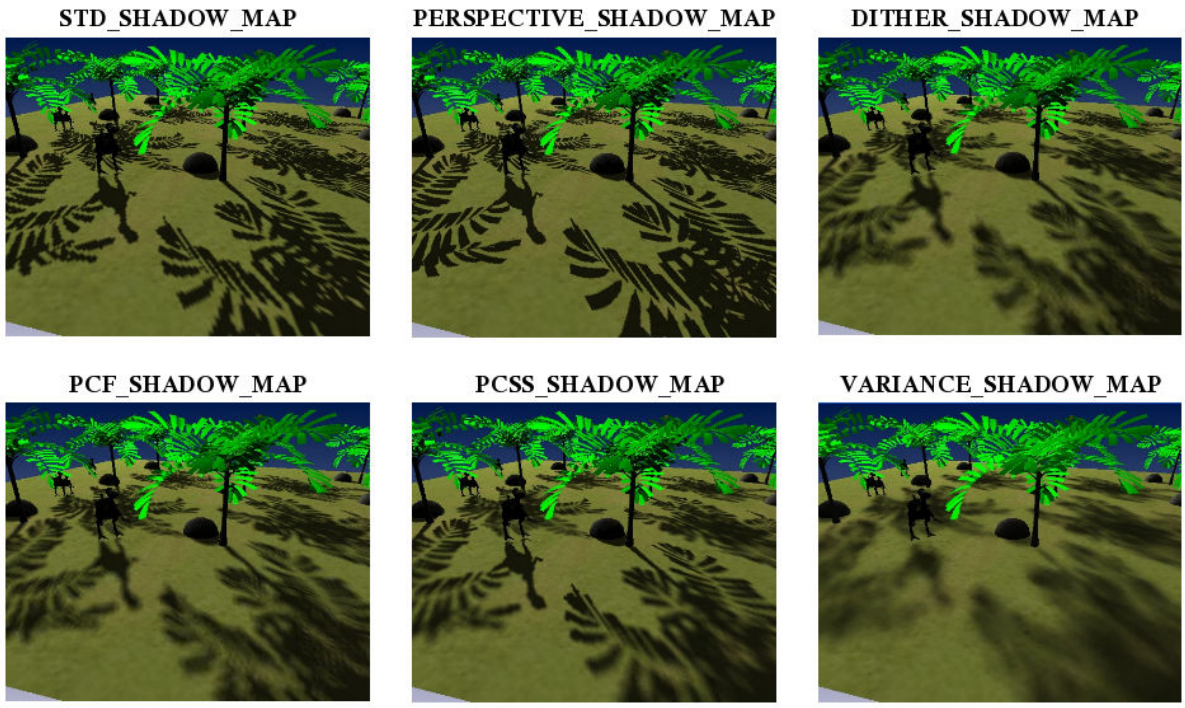

Figure 6.10: *Comparison of different shadow mapping algorithms, which map to shadow modes such as fastHardShadow, niceHardShadow, fastSoftShadow and niceSoftShadow.*[1]

a global shadow intensity and overwrites the per-light settings (if not 0). Again, the range is $[0, 1]$, with higher values being more intense (i.e. darker).

Shadow mapping is particularly tricky to get right when transparency comes into play and impacts on rendering speed. Thereto, the field "shadowExcludeTransparentObjects" is a boolean responsible for the handling of transparent objects like glass etc. Due to the mentioned problem that come along with shadow mapping, there are two more fields which can be used for fine grained control of the shadow mapping implementation: the "shadowOffset" field defines the polygon offset for alleviating artifacts due to depth buffer precision problems. The "shadowMapSize" field defines, as the name implies, the size of the shadow map, which is again quite useful for controlling the speed-quality trade-off for shadow-map based implementations.

The "mode" field allows the selection of different shadow calculation methods, as depicted in Figure 6.10. In our proposal [151], all X3D run-time environments should at least support the following values in order to account for different requirements and application types: "auto", "uniformHardShadow", "perspectiveHardShadow", "uniformSoftShadow", and "perspectiveSoftShadow". Whereas "auto" activates the default browser settings, i.e. no shadows, all other modes are implementation dependent and it can be dynamically switched between them. We only assume that the given order reflects the increasing complexity of the method, "uniformHardShadow" should be the fastest and "perspectiveSoftShadow" should offer the best quality shadows.

[1] Image partially reproduced in [2, p. 368].

In our implementation, for example, we map the previously mentioned PCF-method to the "niceUniformSoftShadow" mode and the modified PCSS [82] to the "perspectiveSoftShadow" modes. In contrast to uniform PCF shadows the latter technique leads to more convincing results, but because of the more complex filtering, in our experiments frame rates dropped to at most 30 to 40 % of those achieved with PCF with a SM 3.0 GPU.

Standard shadow maps correspond to the "uniformHardShadow" mode and light space perspective shadow maps (LISP shadows [345]) are mapped to the "perspectiveHardShadow" mode (see upper left part of Figure 6.10). In our test scenes the frame rates of LISP shadows were only around 7 percent lower than with standard shadow mapping, which is a minor loss compared to the big gain in visual quality.

Different shadow modes are not only useful for quality-speed trade-offs, but also for simulating different types of light sources – a single light bulb throws hard shadows, whereas an area light source such as a window throws very soft shows. A more in-depth discussion of suitable methods including a comparison, implementation details and possible improvements can be found in [124]. The following shadow modes are available in the current implementation (cp. Figure 6.10), whereas all of them are based on shadow mapping, which as discussed in Section 3.2.4 scales best for complex and dynamic scenes:

uniformHardShadow Standard shadow maps.
perspectiveHardShadow Light space perspective shadow maps (LISP).
fastUniformSoftShadow Dithered soft shadows, comparable to the PCF shadows.
niceUniformSoftShadow Percentage closer filtering (PCF).
perspectiveSoftShadow Percentage closer soft shadows (PCSS).
uniformSoftShadow Variance shadow maps.
none Deactivates shadows completely, same as "auto".

We have implemented the described algorithms in OpenSG 1.8x [232], which (as already discussed in Section 5.3.2) does not provide any means for multi-pass rendering. Thereto, we have developed the `osg::ShadowViewport` node that is not only responsible for rendering the whole scene-graph but also for rendering and compositing the different shadow passes. For extensibility the shadow viewport follows a modular design. Each shadowing technique is integrated with an own module that derives from the abstract `TreeRenderer` class. Depending on the chosen shadow mode, an object of the corresponding type (e.g. `PCFShadowMap`) is instantiated, which is visualized in Figure 6.11.

Usability improves much, if shadows can be displayed without the need for modifying already existent shader programs, which normally are responsible for all texture lookups including the shadow maps. In our implementation we solved this problem by also introducing a "shadow factor map", an intermediate texture, which contains the results of the shadow tests. The final shadow composition is done by first rendering the whole scene without additional shadows into a texture called "color map", followed by the "shadow factor map" creation, and finally the multiplication of both.

A quality improvement without noticeable decrease in performance can be gained by filtering this texture [283]. Because simple filtering leads to halo effects around unshadowed objects, which are occluding shadowed regions, another texture containing an edge image from camera view is generated with the help of an edge detection shader. Now, blurring is only done, if the filter region does not contain a silhouette edge. Finally, rendering

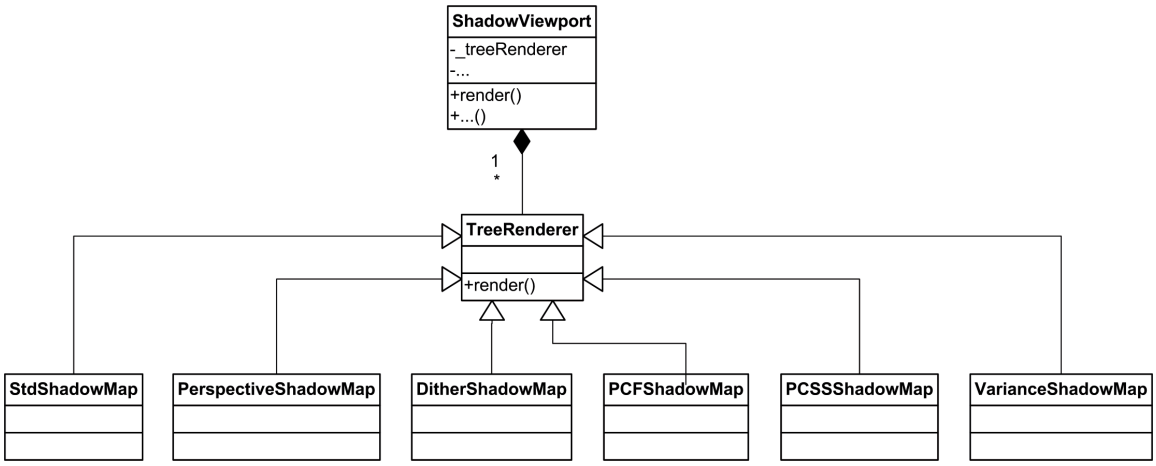

Figure 6.11: *Coarse structure and basic components of the OpenSG ShadowViewport class.*

performance can be further improved when in the shadow pass only depth is written and all other material chunks are switched off, with the exception of special objects like hair, as explained in the last paragraph of section 4.5.4.2, where the final appearance to a great extent is revealed by the alpha map.

6.3.4 Multi-pass Rendering in X3D

Multi-pass can basically be understood in two ways. On the one hand it means the ability to dynamically render a partial scene-graph, which does not necessarily need to be part of the original scene, to an offscreen texture, that can then be used for creating effects like reflection and refraction or shadows. In the Xj3D extension documentation [347] a simplified possibility for creating such offscreen images was first proposed with the *RenderedTexture* node. On the other hand the term multi-pass denotes the ability to render geometry in an ordered sequence, usually with different drawing operations like blending, depth or stencil enabled. A first proposal for providing access to low-level rendering modes in X3D can be found in [26].

6.3.4.1 Layering

AR applications in general require methods that render different layers of information – at least some sort of video-stream as background and 2D- and 3D-annotations. Even more complex layering techniques with compositing methods are needed to implement image-based rendering techniques like the proposed differential rendering algorithms. The current X3D ISO standard does not really support layers but only very simple and rigid background and foreground settings. The *BackgroundBindable* nodes are always sky-boxes and therefore not really useful for our purpose. The image which is synthesized while rendering the scene defines the foreground. No additional layers exist. There are ways to simulate 2D overlays, e.g. heads-up displays (HUD), with the *ProximitySensor* node, but they are very limited and are not image-based methods at all.

The proposed X3D Specification Revision 1 [336] includes two new components, Layering and Layout, which provide nodes and functionality to render and layout different scene-parts in different layers. The *Layer* and *LayerSet* nodes define the sub-trees and rendering order but do not define what kind of composition method is used. The spec only defines that the layers will be rendered "on top" of each other. Layer nodes are intended to create special 2D-/ 3D-interaction elements such as HUDs or non-transforming control elements. With the new *Viewport* node additional clip boundaries can be defined, but they only refer to a single render window.

This gets even worse when having a closer look at the Layout component. Because of nodes like the *ScreenFontStyle* and its pixel-specific addressing it is mainly designed as a means to provide some additional information and with desktop applications and interaction metaphors in mind. Nevertheless, for our proposed environment, both following extensions are essential. We need window-sized and view-aligned quads and additionally some way to control the image composition method used. The first requirement can be fulfilled by using a slightly extended *LayoutLayer* node from the latest specification revision. For the second requirement we introduce novel *X3DAppearanceChildNode* types to control the color-/ stencil-/ depth-buffer writing and merging [151, 156].

These extensions can be used to perform a wide number of different multi-pass methods. Multi-pass techniques in general are not only necessary for differential rendering (page 194) but they are also useful for all image space rendering operations (for example rendering to texture space, and writing normals or depth into a backbuffer or texture for accessing and evaluating e.g. neighboring information in an appropriate shader). In combination with our modified *LayoutLayer* node as described before, it is also a very powerful method for doing post processing effects like blur or gloom well known from games (see Figure 6.4, left, in section 6.2.4 for some examples) [151, 23].

As already mentioned we are using an extended *RenderedTexture* node [347] (see below) in order to provide the ability for offscreen rendering including associated buffers like the depth buffer. Our modified *RenderedTexture* is derived from the *X3DEnvironmentTextureNode* and has an SFBool field called "depthMap", which allows the automatic generation of depth maps for e.g. additional user created shadows. Because this is only useful in combination with appropriate transformation matrices, the projection (modelview projection matrix of camera space) and viewing out-slots (model matrix of parent node) are added.

```
RenderedTexture : X3DEnvironmentTextureNode {
 SFNode     []       textureProperties NULL
 MFNode     []       excludeNodes []
 SFString   [in,out] update       "NONE"
 SFNode     [in,out] viewpoint    NULL
 SFNode     [in,out] background   NULL
 SFNode     [in,out] fog          NULL
 SFNode     [in,out] scene        NULL
 SFNode     [in,out] foreground   NULL
 MFInt32    [in,out] zOffset      []
 MFNode     [in,out] targets      []
 MFInt32    [in,out] dimensions   [128 128 4 1]
 MFBool     [in,out] depthMap     []
```

```
 SFBool     [in,out] readBuffer   FALSE
 SFMatrix4f [out]    projection   identity
 SFMatrix4f [out]    viewing      identity
}
```

The only way to persistently change texture data that is already hosted on GPU memory is to directly render into this texture [157]. Hence the field "targets" can hold references to all kinds of texture nodes including 3D textures. The field "zOffset" defines the z-offset of a slice of the given 3D texture, which allows to render exactly into that layer. To support 3D textures in general, despite the original proposal the "dimensions" field now has four parameters for specifying width, height, depth, and pixel type of the texture. Finally, if "readBuffer" is true, the framebuffer content is read back into the image.

Using offscreen buffers has the additional advantage that the creation of floating point textures can be forced (likewise with a new texture properties field). This not only allows doing shading calculations with higher precision, but also allows HDR rendering, especially in combination with support for special HDR image formats like OpenEXR and Radiance. However, it is not possible to use X3D pointing sensors or even to navigate directly within an offscreen buffer. Thereto, we extended the X3D scripting API by another method *getView()* that can return any type of render surface and on which a ray can be shot for implementing interactions.

Furthermore, for special effects one often just needs a texture containing the framebuffer content. For that purpose we provide the *TextureGrabLayer* node with an *SFNode* field "texture", which – depending on its position in the layers field – simply contains the grabbed framebuffer. Here any *X3DTexture2DNode*, which automatically is resized when the viewport size changes, can be used for later re-USE. An *SFImage* outslot exists but is inappropriate for most applications because in this case the texture first has to be transferred back to the CPU.

```
TextureGrabLayer : X3DLayerNode {
  SFBool [in,out] isPickable FALSE
  SFNode [in,out] viewport   NULL
  SFNode [in,out] texture    NULL
}
```

6.3.4.2 Render State Control

As a requirement, for more complex VR/AR applications the user sometimes needs control over the rendering order of different geometries as well as over low level rendering modes [151, 156, 23]. Currently the only possibility to define at least some sort of rendering order on a per object basis within X3D is by using the *PackagedShader* node in combination with CgFX, D3D9FX or D3D10FX shaders. Despite the platform dependence (for instance the latter only work on Microsoft Windows systems), a *PackagedShader* thereby only defines render states for one single shape [83].

So, for representing the other type of multi-pass rendering, and because the *Appearance* node finally reveals how a rendered *Shape* node looks like, we extended the shape component with some new nodes for setting different render states and therewith the *Appearance*

node with the appropriate fields. Because such functionalities directly map to the API of the graphics board driver, they cannot be encapsulated in X3D protos by the user.

First we introduce the "sortKey" field with $sortKey \in \mathbb{Z}$ for defining the rendering order, what is essential in combination with e.g. color masking and alpha blending, which thereby can be used for generating simple glow effects by rendering another partially transparent object after the original shape. Alternatively, one can think about a special ordering group, but this way usage is much more intuitive and is automatically correct for the whole scene graph. For better readability the X3D "metadata" fields again are omitted in all node descriptions.

```
Appearance : X3DAppearanceNode {
  SFInt32  []       sortKey          0
  SFString []       sortType         "auto"
  SFNode   [in,out] fillProperties   NULL
  SFNode   [in,out] lineProperties   NULL
  SFNode   [in,out] material         NULL
  SFNode   [in,out] texture          NULL
  SFNode   [in,out] textureTransform NULL
  MFNode   [in,out] shaders          []
  SFNode   [in,out] blendMode        NULL
  SFNode   [in,out] stencilMode      NULL
  SFNode   [in,out] colorMaskMode    NULL
  SFNode   [in,out] depthMode        NULL
  SFNode   [in,out] faceMode         NULL
}
```

For rendering operations, which belong closely together as it is the case for the two pass hair shader mentioned in section 4.5.4.1, or even for a simple toon shader for non-photorealistic rendering, we also introduce the *MultiPassAppearance* node as the generalized extension for the X3D *TwoSidedMaterial* node [151, 156]. The "appearance" field simply contains an ordered sequence of single pass appearance nodes. Although the *PackagedShader* is especially intended for this usage by providing an interface to effects languages like CgFX or HLSL FX, it is only suited for specific target platforms, as opposed to the more generically designed *MultiPassAppearance*.

```
MultiPassAppearance : X3DAppearanceNode {
  SFInt32  [in,out] sortKey    0
  SFString []       sortType   "auto"
  MFNode   [in,out] appearance []
}

AppearanceGroup : X3DGroupingNode {
  SFBool [in,out]  render     TRUE
  MFNode [in,out]  children   []
  SFNode [in,out]  appearance NULL
}
```

Additionally we propose an *AppearanceGroup* node which extends the *Group* node with an "appearance" field. This is quite useful if a whole group of *Shape* nodes, like head, eyes and hair, should share the same material properties as is the case for the light pass mentioned in chapter 5.1.1, where the fragment's distance to the light source is written and fragments with alpha values smaller a certain threshold are discarded.

In the following, a few nodes for allowing finer control over the rendering modes are shown. If the corresponding fields in the *Appearance* node are not set by the user, the X3D browser uses standard settings which fit best for the current material, texture or shader. Otherwise the state modes, for example the *StencilMode*, override the default settings. A short outline of the proposed node interfaces shall conclude this proposal.

```
BlendMode : X3DAppearanceChildNode {
  SFString [in,out] srcFactor         "one"
  SFString [in,out] destFactor        "zero"
  SFColor  [in,out] color             1 1 1
  SFFloat  [in,out] colorTransparency 0
  SFString [in,out] alphaFunc         "none"
  SFFloat  [in,out] alphaFuncValue    0
}
```

As the name implies, the *BlendMode* node allows access to blending and alpha test. The field values, for instance "src_alpha" and "one_minus_src_alpha" for standard alpha blending, map directly to the corresponding rendering state names. The fields "alphaFunc" and "alphaFuncValue" specify the conditions under which a fragment is drawn or discarded. With "alphaFunc" set to "lequal" and a given reference value c the fragment passes if the incoming alpha value is less than or equal to c. Concerning choice and naming conventions, a common subset of the OpenGL and DirectX graphics standards, which have already been very well documented (e.g. [174]), was chosen.

```
ColorMaskMode : X3DAppearanceChildNode {
  SFBool [in,out] maskR TRUE
  SFBool [in,out] maskG TRUE
  SFBool [in,out] maskB TRUE
  SFBool [in,out] maskA TRUE
}
```

The other four modes, *StencilMode*, *ColorMaskMode*, *DepthMode* and *FaceMode*, are likewise almost self-explanatory. The *ColorMaskMode* permits control over color masking (the color channel is written if the corresponding mask field is true). With help of the *DepthMode* depth functions can be set, which is especially useful in combination with the previously introduced "sortKey" field and the "solid" field of the *X3DComposedGeometryNode*. The *FaceMode* can be explained best as being a generalized extension of the "ccw" and "solid" fields of the *X3DComposedGeometryNode*, as is needed in combination with the "sortKey" field or the *MultiPassAppearance*.

```
DepthMode : X3DAppearanceChildNode {
  SFBool   [in,out] enableDepthTest TRUE
  SFString [in,out] depthFunc       "none"
  SFBool   [in,out] readOnly        FALSE
  SFFloat  [in,out] zNearRange      -1
  SFFloat  [in,out] zFarRange       -1
}

FaceMode : X3DAppearanceChildNode {
  SFBool   [in,out] smooth       TRUE
  SFString [in,out] cullFace     "auto"
  SFString [in,out] frontFace    "auto"
  SFString [in,out] frontMode    "auto"
  SFString [in,out] backMode     "auto"
  SFFloat  [in,out] offsetFactor 0
  SFFloat  [in,out] offsetBias   0
}
```

The *StencilMode* mentioned in section 6.4.5 permits fine grained control over stencil bit masks and functions. The stencil test thereby conditionally eliminates a fragment based on the outcome of a comparison between the value in the stencil buffer and a reference value [286]. For all fields with default values equal to -1 or "none", implementation specific default values are used. In addition we have extended the *Background* nodes such that they can clear all stencil bitplanes.

```
StencilMode : X3DAppearanceChildNode {
  SFString [in,out] stencilFunc    "none"
  SFInt32  [in,out] stencilValue   0
  SFInt32  [in,out] stencilMask    0
  SFString [in,out] stencilOpFail  "keep"
  SFString [in,out] stencilOpZFail "keep"
  SFString [in,out] stencilOpZPass "keep"
  SFInt32  [in,out] bitMask        -1
}
```

With the help of the proposed node extensions, complex multi-pass appearances as needed for advanced rendering techniques, such as hair rendering as discussed in section 4.5.4.1 or differential rendering as outlined in section 6.4.5, can be created in X3D. Moreover, our extensions of the *X3DAppearanceChildNode* are also useful for other applications (such as for instance image based CSG operations). As is obvious, these types of pixel operations inherently require being able to set the sorting order.

6.4 Mixed Reality

The fusion of real and virtual worlds is the foundation for a range of computer graphics applications: complex Augmented and Mixed Reality, movie effects, and the ability to

Figure 6.12: *Virtual Herr Kaiser in meeting room no. 140 of Fraunhofer IGD.*

advertise products not yet completed, such as houses still being built or prototypes of cars in real environments. To achieve a "mixed" reality feeling, complex lighting interactions between real and virtual objects have to be simulated in real-time. Only if the user is not able to clearly distinguish between real and virtual objects, this aim is reached.

Nowadays, more and more Mixed Reality applications are emerging such as AR/MR maintenance scenarios [262]. Like dialog systems they typically consist of several stages. An example is outlined in section 6.4.6.3. Further, especially in mobile computing in combination with geolocation-based services, there is a recent trend in augmenting the real world with virtual information, which is made possible due to the increasing processing power, bandwidth, and 3D-capabilities even on mobile devices. Thereby, new user interfaces become possible, where e.g. a virtual character, as an augmented master teacher, mediates procedural knowledge like how to use a new device. Hence, there also exist approaches for integrating virtual characters as human-computer interface, because such high-level context elements can more efficiently cope with augmented physical environments than explicit direct manipulation techniques via pointing devices or multi-touch.

Hence, Barakonyi and Schmalstieg [19] proposed a framework where virtual agents with autonomous behavior are employed as interface and interaction metaphor in the context of AR applications. One demonstration scenario was a machine maintenance application, where a virtual animated repairman assisted an untrained user to maintain a real toy robot. A contribution of their work was the implementation of the reasoning mechanism following the BDI (belief-desire-intention) model, an architecture common in Artificial Intelligence to realize the brain of an intelligent agent with its beliefs about the world, its programmed goals, and current plans [214]. This thereby results in certain animated behaviors that finally can be migrated between different AR applications.

Rendering aspects as addressed in the first paragraph and discussed in Section 3.3 were not yet considered here, although they can considerably improve the final results as can be seen in Figure 6.12, where the virtual character is occluded by real scene geometry and moreover throws shadows onto the scene. Thus, in [151, 89, 88] we explored and discussed X3D as an application description language for advanced Mixed Reality environments.

The semantics of the X3D ISO standard describe an abstract functional behavior of time-based, interactive 3D, multimedia information. It is independent of any specific software or hardware setup, and X3D has been established as an important platform for today's web-based visualization. However, X3D clients and applications are mainly built for desktop systems running a web-browser. Yet, there are very few examples for AR systems utilizing X3D beyond a simple geometric description format. In order to fulfill the im-

Figure 6.13: *Comparison: Real lightprobe (left) and rendered sphere with irradiance map.*

age compositing and synthesis requests of modern Augmented Reality applications, we propose extensions to X3D, especially with a focus on lighting and realistic rendering.

6.4.1 Lighting Reconstruction

A major problem with X3D's applicability in the MR domain is the missing support for global illumination techniques. Since for seamless integration virtual objects have to resemble reality as close as possible, simple lighting models won't suffice in this context. In this section we shortly describe the irradiance mapping technique as well as ambient occlusion for enhancing visual quality towards more realism.

6.4.1.1 Irradiance Mapping

The first important step to enable high quality rendering of real and virtual lights is the lighting reconstruction phase, where real world lighting is captured to transfer effects from the real scene onto a virtual model. For instance, a light-switch could be toggled in the real environment, which should have an effect on the virtual objects, otherwise it will be clear rather soon that they are just an augmentation. For static configurations, the incident radiance can be captured with a light probe. Dynamic scene lighting can be captured with a 180° fish eye lens. The acquisition should be done in HDR (high dynamic range [61]) to capture the real radiance, otherwise lighting mapped for different materials will appear to have no contrast.

For mirror-like surfaces simulating complex lighting behavior with indirect illumination has been solved through standard environment mapping techniques. A fish-eye image or a reflective surface respectively, most often a simple sphere, is captured and directly applied to another more complex surface, for instance via sphere mapping. While this usage introduces some errors, at least for different shapes, the results are optically pleasing and usually feel right. But it should be noted, that this only captures the far field.

Our simulation [151] relies on image based lighting, a method that derives all information about the environment lighting from images. Usually, these images are cube- or sphere maps and can be used to simulate highly reflective or mirror-like surfaces. Unlike "simple" environment mapping, the idea of irradiance mapping is to further extend this method.

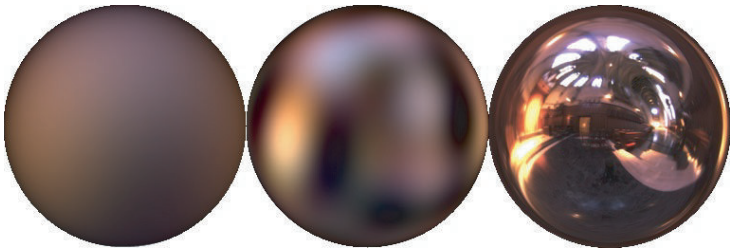

Figure 6.14: *Debevec's grace probe (right) reconstructed with 4 and 11 coefficients (left).*

Instead of just calculating textures for reflective materials, others – for instance diffuse or specular surfaces – can have their own textures as well.

Collectively, these textures are called irradiance (environment) maps. A great deal of research has been invested in finding out how to generate irradiance maps effectively and without much deviation from real light-integration through spherical harmonic analysis. Likewise more recent papers have shed light on base functions that enable quick analysis of different ranges of frequencies [223]. In [256] the authors showed that low-frequency representations of spheremaps can be efficiently calculated with spherical harmonic analysis (see equation 3.14, p. 75).

Currently we use the spherical harmonic basis to simulate diffuse and low order glossy irradiance, because high frequencies are captured inadequately. In our simulation, the reconstruction process has been implemented in a shader, which simply dots the constant spherical harmonic values of the lighting with the corresponding coefficients. The function values of $y_l^m(\theta, \phi)$ (see Figure 3.5, page 75) are precomputed and stored inside textures, which are used within the shader program to determine the values of the spherical harmonic functions. Another implementation can be found in [179]. In order to make these irradiance textures (which have to be regenerated whenever the input image changes), readily available, we propose the *SphericalHarmonicsGenerator* node that will be described in detail later.

By representing a sphere map as a parameterized function, it can be analyzed and filtered to create other irradiance maps. In Figure 6.14, the famous "Grace Cathedral" light probe from Paul Debevec [59] has been filtered to lower frequencies. The original image on the right is used to simulate highly reflective material, whereas the sphere maps in the middle and left represent glossy and diffuse surface appearances. In Figure 6.13, a direct comparison can be seen between the real light probe that was used to capture incoming light from a real scene and a slightly filtered irradiance map applied to a virtual sphere.

6.4.1.2 Ambient Occlusion

Using precomputed radiance transfer, more complex transfer functions can be simulated. Additionally to the irradiance map L, the surface's transfer function T is moved to frequency space (i.e. two harmonic analysis are required). By then exploiting $\int_\Omega L \cdot T \approx \sum_i L_i \cdot T_i$ (which is evaluable inside a shader program), the integral of the rendering equation is approximated by the dot product of these coefficients and the coefficients of the irradiance map while retaining high rendering speed.

Figure 6.15: *Left: ambient occlusion (AO). Right: AO and irradiance mapping combined.*

But PRT is limited to rigid bodies only since the cost of analyzing transfer functions in real time is still too high. Methods like LDPRT [289] can cope with this problem by concentrating on local effects, though leaving distant effects like shadows aside. Other approaches [74] directly tackle deformable objects like virtual humans but rely on heavy precomputation and manual work, why here integrating PRT is left as future work.

The perhaps most obvious difference between local and global illumination are shadows, especially self-shadowing. Without, objects seem to have no detail in structure, but with self-shadowing, tiny structures become emphasized and add to the overall realism. Many of those details can be recovered with ambient occlusion (AO) [245], which describes the statistical light incidence and corresponds to shadows by fully diffuse illumination.

We use AO as a substitute for the transfer function in PRT [151, 89], because it is perceptually one of the most obvious factors and for fewer samples it can be calculated in real-time also for non-rigid objects such as animated characters. For high-polygon models or rigid objects, we precompute and store the same information along with its colors. Alternatively, this step can be done during initialization. Before the irradiance maps are applied to the objects surface, the AO values for all vertices are determined through a modified version of the method described in [271]. Likewise, for good performance we employ the occlusion query OpenGL extension [8].

Real scenes contain much indirect lighting, so only using direct light sources as sampling positions is unrealistic. Hence, random sample positions are used to determine the ambient occlusion on the model. In Table 6.1, we have measured the difference between the generated AO values of a 1000 sample reference model and the same model with a lower sample count. One can see that even for complex geometry, the error drops below 10% with 25 samples. The relative difference with 5-sample-steps drops below 1% at 50 samples for most models. Combining both methods, the colors from the irradiance maps (Figure 6.14) are then attenuated with the ambient occlusion values on the surface, as exemplarily shown in Figure 6.15.

Samples	EG Dragon Smooth	EG Dragon Normal	Buddha
5	21.6528 %	16.6647 %	24.1992 %
25	9.50936 %	5.29504 %	8.72107 %
50	7.32016 %	2.94598 %	6.15287 %
100	4.02114 %	1.42048 %	4.46811 %
150	3.33963 %	2.80035 %	3.65112 %

Table 6.1: *Statistical deviation from converged ambient occlusion depending on smoothness of surface (compare column 1 and 2).*

6.4.2 Material Reconstruction

To accurately map virtual light or shadows onto real surfaces, their material properties have to be known upfront. For instance, to simulate the interaction between a virtual light and a real surface it has to be clear whether or not that surface is diffuse or mirror-like. If these properties are unknown, the differential rendering (see section 6.4.5) will produce wrong colors for shadows and lights or other artifacts (compare lower left part of Figure 6.17). Hence, material properties like the diffuse base color or more complex BRDFs need to be derived from the original footage. Thereby the pixel values in the footage are treated as function values $L_o(x, \vec{\omega_o})$ of the rendering equation (cf. section 3.2.2):

$$f(x, \vec{\omega_i}, \vec{\omega_o}) = \frac{L_o(x, \vec{\omega_o})}{L_i(x, \vec{\omega_i}) \cdot (\vec{\omega_i} \cdot \vec{n}) \mathrm{d}\vec{\omega_i}} \tag{6.1}$$

One problem is the dependence on the geometric scene information, which in the following we assume as given. In our simulation [89], an off-line process tries to analytically estimate material properties from camera images. Because the footage usually does not cover the whole hemisphere for all surface points, this process is a modified implementation of [97] to derive complex materials from the original scene radiance given by HDR images. Combined with real lights in the image, placeholders for unknown lights, so-called virtual lights, are adapted to match the irradiance of the surface.

Diffuse materials can then be estimated iteratively with a linear equation system. As soon as non-linear components are added to the surface BRDF, other solutions have to be found. The authors proposed minimizing a cost function with non-linear optimization for all unknown variables. Instead, we used genetic algorithms for a unified approach to recover all material functions and to alleviate the non-linear optimization problem. As outlined next, a fitness function determines the error between the radiance of the derived BRDF and the surface samples.

6.4.2.1 Genetic Algorithms

A genetic algorithm is a particular class of evolutionary algorithms that is used for global search and optimization problems. Instead of calculating in a deterministic manner a result is evolved from a population of possible solutions. The main motivation for genetic

algorithms as a substitute to approximate surface reflection functions in our implementation (which uses the genetic algorithm C++ library GAlib [328]) is that they require no knowledge of the problem-space. Therefore, one single implementation is sufficient to estimate unknown variables for all kinds of BRDF's. Likewise, very recently genetic algorithms were utilized for finding the location of light sources [63].

We encode all variables in a simple vector, which simultaneously serves as a genome [89]. To evaluate the fitness of a possible result, a cost function simply generates values i_v for all visible pixels of the surface using the evolved genome. All lights, including virtual lights, are taken into the equation. All generated pixel values are then subtracted from the pixel values i_r of the real surface in the photograph. The fitness $q \in (0, \infty)$ of a genome can be calculated with $q = \frac{1}{\sum |i_v - i_r|}$. In the unlikely case that the denominator equals zero, a perfect match (i.e. genome) has been found.

For generating e.g. Figures 6.20 and 6.21, diffuse materials were reconstructed through the iterative method described above. We have tested a steady state genetic algorithm on a simulated Phong material f to evaluate the quality of a reconstruction. 1000 surface samples were gathered to determine the parameters ρ_d, ρ_s and n, with a population size of 100 genomes. Test results point out that linear parts of $f(x, \vec{\omega_i}, \vec{\omega_o}) = \frac{\rho_d}{\pi} + \frac{n+2}{2\pi}\rho_s \cos^n \gamma$ were evaluated with less deviation from the actual parameters than non-linear parts. While ρ_d was evaluated correctly in most cases, i.e. no mutant or local minima, the deviations in ρ_s and especially in n for the specular term were generally too high.

It is still unclear whether a larger population or higher mutation rates will lead to better results, which is left for future work. Also, more dense sampling results in better approximations, yet the deviations in non-linear factors are still unacceptable. It should be noted that these test cases exclusively deal with known BRDF's and do not contain any lighting information from the image, neither virtual nor real lights. In the actual implementation, the process iteratively factors out virtual light sources. Ultimately, the calculation of the BRDF parameters that follows this estimation is replaced by the genetic algorithm.

6.4.3 Parameterizing Shadows

For lighting configurations that can be represented as images or spherical functions, the subsequent harmonic analysis can produce sets of spheremaps to simulate the reflection of light on the surface of an object. This method allows for high quality rendering and lighting simulation – a task which is crucial for MR applications. Since the frequency representation of an irradiance map does not include transfer functions on the model's surface other than basic reflection behavior (diffuse, glossy, reflective) nor any drop shadows the model might cast, those have to be included by other means, e.g. Precomputed Radiance Transfer [288]. PRT however is limited to rigid objects.

To solve this problem, we have used high quality shadow algorithms like PCF or PCSS [151], which are described in paragraph 6.3.3, in conjunction with a modification of real time ambient occlusion methods [271] as a substitute for more complex transfer function properties. While this approach does not address special surface or material properties like subsurface scattering, it yields better visual quality than the simple lighting models of today's graphics hardware while retaining high rendering speed. High dynamic range

Figure 6.16: *Different materials and indirect lighting simulated through irradiance mapping.*

irradiance maps further help to add realism, thus making a clear distinction between real and simulated objects harder than with simple rendering techniques.

One problem is that irradiance mapping does not address any positional information about light sources, whereas ambient occlusion is not considering light sources at all. Without this information, casting shadows into the right direction will become difficult. Depending on the application and the available data, for maximum scalability we have considered on- and offline methods for our simulation.

On the one hand, in section 6.3.2 a technique to extract light sources at runtime is outlined. And on the other hand, it is also possible to first extract direct light sources from images in a pre-process. While the direction can be taken directly from the irradiance map, the position is determined by intersecting its boundary points with the surrounding reconstructed scene model. The extracted positions are then used to project shadows onto geometry. Visually pleasing results were achieved with PCF and PCSS shadows, although the latter caused some performance hits.

To determine the shadow's intensity during runtime, we used the first coefficient of the SH analysis of the surrounding lighting configuration. Because the first SH function is constant, the first coefficient will statistically provide information about the ambient brightness in the scene. Thus, the inverse value (given that c_0^0 is normalized to $[0, 1]$) can be used as shadow intensity. The brighter the ambient lighting is, the less intense the shadow will be and vice versa. The same value can be used to adjust the AO values.

6.4.4 The SphericalHarmonicsGenerator Node

For maximum performance the function values $y_l^m(\theta, \phi)$ of the real valued Legendre polynomial, being used as the (orthonormal) base functions, are pre-computed and stored in a 3D texture, which is afterwards used as a lookup map on the GPU. Here, l is the band index, and m determines the polynomial within band l. For fast de-convolution the reconstruction of the signal f is done in a shader that summates the products of the constant SH values with the corresponding coefficients c_l^m as shown in equation 6.2.

$$f(x) = \sum_{l=0}^{n} \sum_{m=-l}^{l} c_l^m y_l^m(x) \tag{6.2}$$

The *SphericalHarmonicsGenerator* node is a special texture node that generates an irradiance map from another (sphere-) map that is given as input texture. Furthermore, with

this node we introduce the concept of *DependentTextureGenerator* nodes to X3D, which are textures, whose pixel-values depend on and are controlled by other textures [151, 88]. Another example node that also inherits from this base node type is the *NormalMapGenerator* [135] that creates a normal map from a given bump map.

Irradiance mapping [256] allows the creation of low-frequency representations of spherical images (here it was also shown that diffuse materials only require the first 9 coefficients), which in turn can be converted back to textures. These textures can later be used just like usual environment maps to simulate low-frequency reflection on diffuse surfaces.

Additionally, the *SphericalHarmonicsGenerator* calculates a set of spherical harmonic coefficients. Thereby, this new irradiance map can then be used, via sphere mapping, to simulate reflected light from an object with a certain material type. The original spheremap is a transformed fisheye camera image or a quadratic image of a mirror-like sphere and contains the reflection of the surrounding scene. The corresponding parameter is called "irradianceMap". The texture node *irradianceMap* serves as input to the calculation, which can internally be analyzed either by unbiased samples or importance sampling for calculating the SH coefficients.

Because spherical harmonics are used to analyze distant lighting on a hemisphere, the texture is usually a spherical image of a certain parameterization, which is set via the field *mapMode*. This parameter can be either "latLong" (latitude-longitude maps as already discussed in section 6.3.2), "sphere" for sphere maps, "cube" for cube maps, or "auto" for a default mode, in order to change the internal behavior of how to access a given texture map to gather samples for a given direction (θ, ϕ). Direct3D also provides a function call[2] to convert cube map textures into three spherical harmonic coefficient vectors of a given order, one for each color channel RGB, which probably could be further generalized for integration into X3D.

```
SphericalHarmonicsGenerator : X3DTextureNode {
  SFNode   []       textureProperties NULL
  SFNode   [in,out] irradianceMap     NULL
  SFString [in,out] mapMode           "auto"
  SFInt32  [in,out] numBands          3
  SFInt32  [in,out] numSamples        1000
  MFVec3f  [out]    coefficients_changed
  SFFloat  [out]    ambient_changed
}
```

Two other parameters are needed for the spherical harmonic implementation to work. For one thing we have the spherical harmonic order of the analysis, or the number of bands l that will be integrated respectively. This information determines the type of irradiance map that will be generated and is identified by the parameter "numBands". For another thing we have the number of samples that are used to do the Monte Carlo integration, which is identified by the parameter "numSamples", and which can be further used to gain control over matters of speed versus quality of the internal Monte Carlo integration.

[2] D3DX Functions (Microsoft): http://msdn.microsoft.com/

As soon as the *SphericalHarmonicsGenerator* node has finished calculating the coefficients, they are available through the "coefficients_changed" slot, a vector of type MFVec3f and length l^2, where each SFVec3f element carries a coefficient for each color channel RGB. Although these coefficients can be directly used for determining the radiance transfer as described in section 3.2.2, here they are currently only used for low-pass filtering the irradiance map to obtain diffuse lighting, because, as the name implies, PRT requires heavy precomputation and is not suited for non-rigid objects like virtual characters. Further, to also simulate the rotation of a rigid object, in [88] an X3D matrix node extension is proposed for rotating a spherical harmonic coefficient vector.

Another slot, which has been of special interest to our mixed reality implementation, is called "ambient_changed". As mentioned in the previous section, the very first coefficient is computed with the first spherical harmonic function, which is constant. Therefore, the coefficient can be used to determine the ambient brightness analyzed texture *irradianceMap*. This has one major application: to adjust the intensity of shadows, the overall brightness inside a scene has to be known upfront.

If the surrounding scene is very bright, the statistical intensity of the shadow is low and vice versa. The slot returns a value scaled to $[0, 1]$ whenever the ambient brightness changes, with 0 being totally dark and 1 being totally bright. This value can then be used to adjust the shadow intensity for standard shadow mapping algorithms by routing it to the new "shadowIntensity" field of a *Light* node as described in section 6.3.3.

6.4.5 Differential Rendering

As already stated, a reconstruction of the real geometry, for which we used an external tool and which is out of scope here, generally is necessary at least for handling occlusions and receiving shadows. Especially for AR-supported assembly simulations and the like correct occlusion handling is considered an important visual clue: e.g., for the inexperienced user it makes a big difference, if a virtual screwdriver appears behind or in front of the real workpiece. As mentioned, differential rendering is a multi-pass compositing technique that is feasible for augmenting images or videos with consistent illumination. It requires two lighting simulations, one with the real scene only and a second one with the additional virtual objects inserted. For real-time applications the rendering should be hardware accelerated, therefore both before mentioned scenes are rendered into different textures using standard rasterization methods.

Because two lighting simulations or respectively rendering passes are needed for calculating the difference image (see Figure 6.17 for visualization), we first need to introduce the concept of multi pass rendering in the context of X3D, which was discussed in paragraph 6.3.4. Because occlusions must be handled correctly beforehand, for doing differential rendering the virtual objects need to be rendered after the real scene, with the stencil test function set to "always". This way a mask is created at only those pixel positions ultimately containing parts of the virtual object despite the original order of incoming fragments. Therefore a fine grained control over rendering order and rendering states is needed, which is discussed in section 6.3.4.2.

After that we can go into details. Let L_{orig} be the original scene radiance provided by

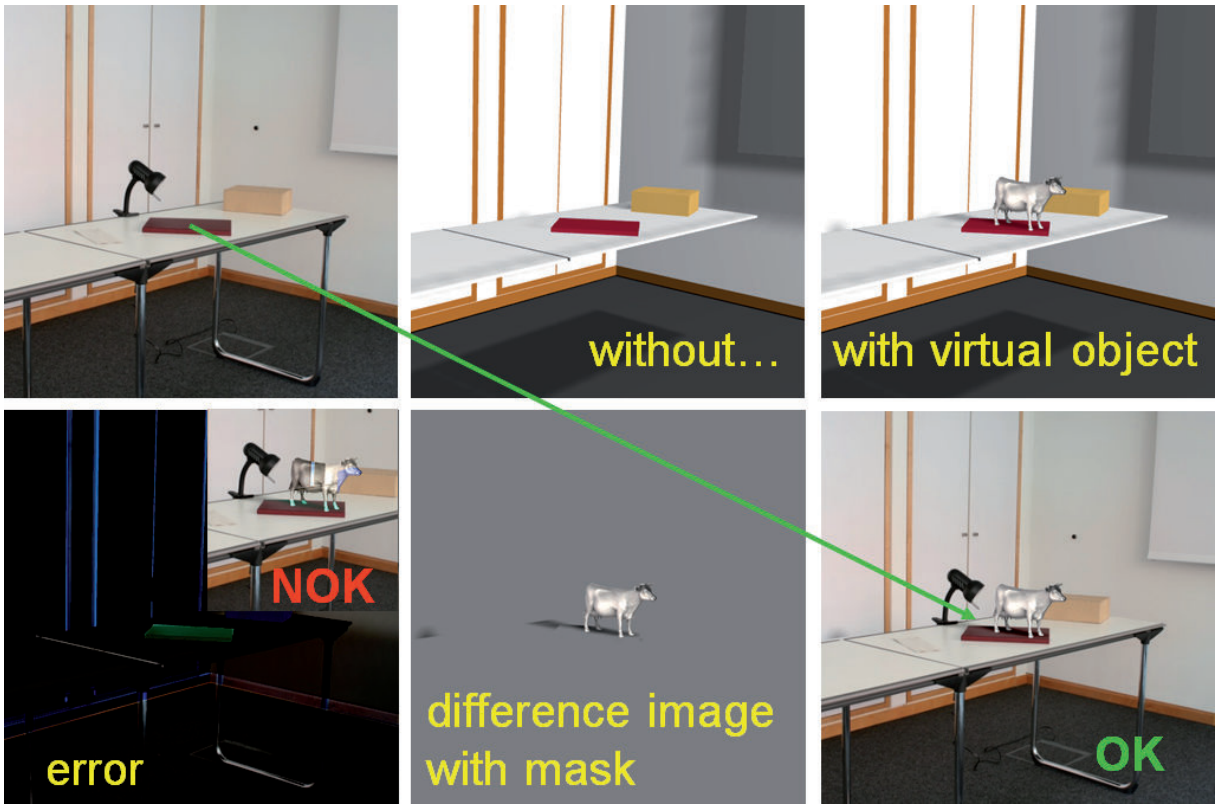

Figure 6.17: *Steps in differential rendering (from left to right): Original image, reconstructed scene without virtual object, scene with virtual object, visualization of error ΔL_{err} and unsuitably big error, difference image with mask (for illustration purposes an offset is added, so that zero is grey), final augmented image.*

the background image, L_{with} the appearance with the virtual objects inserted and $L_{without}$ the appearance without the virtual objects. Then the error in the rendered scene without synthetic objects compared to the real image is $\Delta L_{err} = L_{without} - L_{orig}$. An alternative approach employs the relative error for avoiding possibly negative results. As can be seen, the better geometry and material reconstruction are, the smaller the resulting error is. By subtracting the error from L_{with}, the changes in illumination caused by inserting the virtual object (which are e.g. forming shadows if L_{with} is less than $L_{without}$) can be represented as follows:

$$\begin{aligned} L_{final} &= L_{with} - \Delta L_{err} \\ L_{final} &= L_{orig} + (L_{with} - L_{without}) \end{aligned} \tag{6.3}$$

By using our modified *RenderedTexture* node for rendering the different scenes, and shadow generation turned on, the values for L_{with} and $L_{without}$ are obtained for all corresponding pixels of the original image. In combination with our slightly modified *LayoutLayer* node for simplifying the rendering of a window-sized view-aligned quad with arbitrary appearance – a requirement well known from computer games for creating special visual effects by means of a post-processing step in image space – the combination of

```
1  uniform sampler2D objScene;
2  uniform sampler2D scene;
3  uniform sampler2D img;
4
5  void main()
6  {
7    vec4 with    = texture2D(objScene, gl_TexCoord[0].st);
8    vec4 without = texture2D(scene, gl_TexCoord[0].st);
9    vec4 orig    = texture2D(img, gl_TexCoord[0].st);
10   gl_FragColor = orig + with - without;
11 }
```

Listing 6.1: *GLSL fragment shader code for the final compositing of all image layers.*

all renderings to form the final output image is now very easy [151]. For maximum performance in an after effects layer these three textures are combined in the GLSL fragment shader program shown in Listing 6.1.

Because reconstruction is always inaccurate (especially concerning the real material properties as outlined in paragraph 6.4.2 – see bottom left image in Figure 6.17) and hence $\Delta L_{err} \neq 0$, in the final imaging layer, the texture containing the real scene with the augmentation is rendered in such a way that it simply replaces the virtual objects by means of a stencil buffer mask by setting the stencil function in the *StencilMode* node to "keep".

6.4.6 System Setup

The Mixed Reality simulation setup is shown in Figure 6.18. We use a lightprobe or a fish eye camera to capture the real lighting configuration and a second (preferably HDR) camera for the scene. Together with the reconstructed geometry and material information about the real scene, the virtuel object is fed to the simulation that merges the real world with the virtual object via differential rendering (see section 6.4.5). The composition via differential rendering is then displayed on a screen.

6.4.6.1 Integration of Tracking Algorithms

This section very shortly outlines how an MR application can be developed by using the previously proposed techniques together with the Instant Reality framework [135]. Therefore, Figure 3.8 (on page 81) visualizes the connection between the tracking and the rendering system in Instant Reality, which is handled by the "InstantIO" subsystem [51]. Figure 6.19 (left) shows an application using the *Viewfrustum* node in combination with a *DirectSensor* node for augmenting the real model with a virtual flow field running at interactive frame rates.

The left window shows the tracking view used for data integration. The tracking system which we connected via the mentioned data stream sensor nodes [52, 151] was developed by members of our group and uses an approach based on a combination of line tracking for initialization (which internally also needs a geometric model of the real scene), and KLT point features for frame to frame tracking as described in [27].

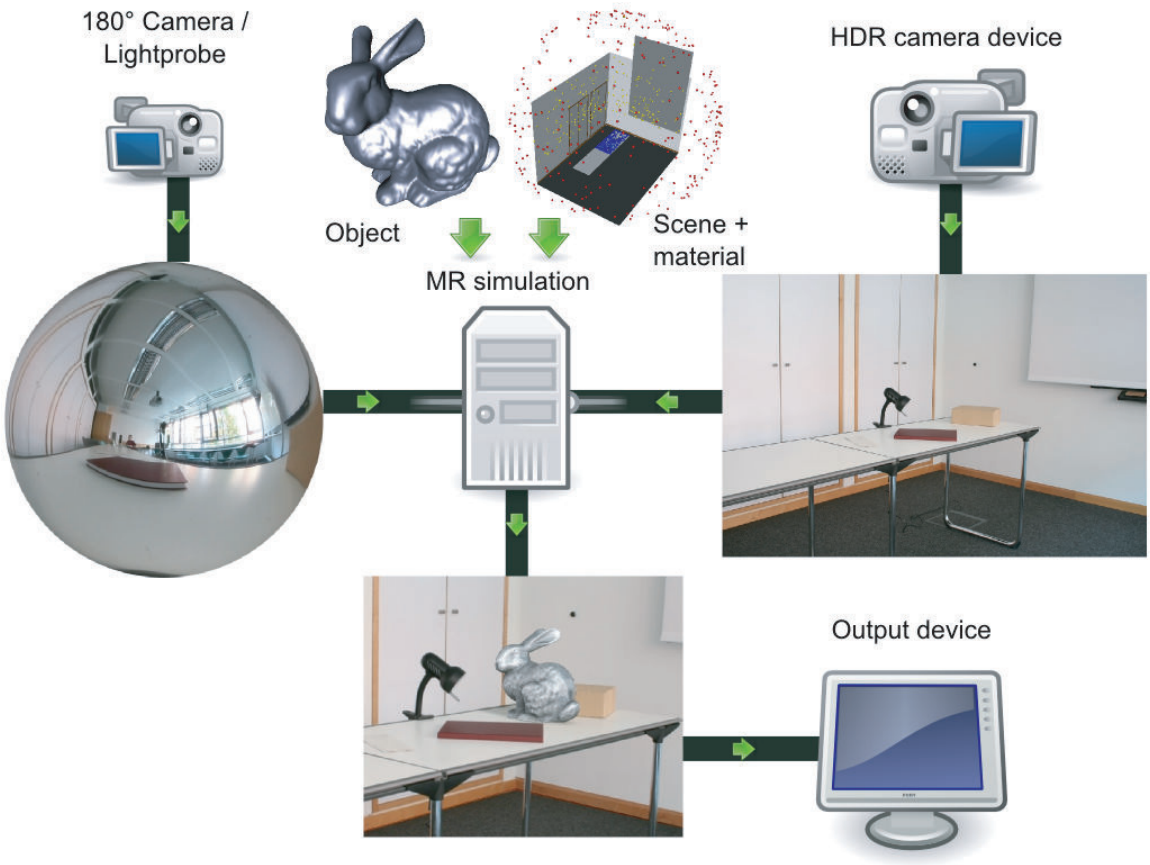

Figure 6.18: *The system setup of the described Mixed Reality simulation.*

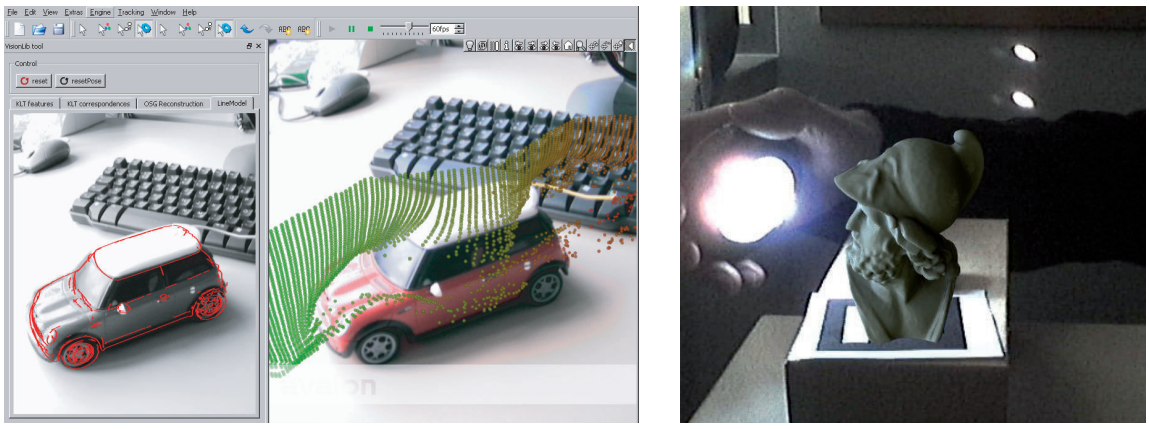

Figure 6.19: *Left: A toy car augmented with a dynamic flow field (the leftmost image shows the tracking view). Right: An X3D-based Mixed Reality simulation with relighting.*

Figure 6.20: *The Stanford dragon model in the entrance hall of Fraunhofer IGD.*

In Figure 6.19 (right), the spherical environment images needed for lighting are provided by a fish-eye camera inside the little box below the head model. Another camera additionally films the entire scene, into which the object is merged with the help of a tracking algorithm (in this case marker tracking) to control position and orientation of the *Viewpoint* node. Here, the X3D scene consists of two different layers, the video in the background and the 3D reconstruction of the bust.

6.4.6.2 Rendering Results

A standard Intel Pentium IV @ 2.4 GHz PC with a NVidia 6600 GT graphics board and 1 GB memory was used to conduct our tests, with the Instant Reality framework [135] for application description that utilizes OpenSG [232] for rendering. Test data was acquired with a Canon EOS 350D camera for scene backgrounds and the light probe. For creating Figures 6.20 and 6.21, the HDR photos were generated via Debevec's HDRShop [308].

The image displayed in Figure 6.20 combines all techniques presented here in one X3D application to merge virtual and real environments. The well-known dragon model is rendered in a 1500×1000 pixel context with $8 \times$ FSAA, a spherical harmonic analysis of the incoming lighting data with 9 coefficients, a mixture of 25% diffuse and 75% specular lighting resulting from the HDR irradiance map, static ambient occlusion and uniform PCF shadows as discussed in section 6.3.3.

The PCF shadow intensity is set via the "ambient_changed" outslot of the *SphericalHarmonicsGenerator*. The image is afterwards combined with the real background image through differential rendering and drawn at ten frames per second. Much higher frame rates (up to 60) are achieved for low-polygon models such as the Stanford bunny. For less complex models dynamic ambient occlusion can be enabled without major performance hits, though depending heavily on the sampling rate.

The differential rendering automatically handles occlusions from real objects to virtual

Figure 6.21: *The Stanford dragon in a real scene between a table-leg and a carton. Shadows and occlusion are handled via differential rendering and reconstructed geometry.*

ones or vice versa, as shown in Figures 6.12 (where a virtual character is inserted into the real meeting room scene) and 6.21. To assure that light and shadows are transfered correctly onto real materials, the material reconstruction as described in section 6.4.2 is used to gather information about the surface the object is placed on.

6.4.6.3 Exemplary Use Case: VR/AR Manuals

In the industrial field each domain handles a variety of devices and machines. Often people who are not experts have to first find the appropriate manual and then try to understand the diagrams and textual explanations to repair a machine by themselves. This results in two issues, which can be solved by Virtual and Augmented Reality technology. On the one hand the huge number of manuals can be replaced by a database of electronic instructions that can be searched more efficiently, and on the other hand schematic drawings of work steps to perform become more descriptive due to real images of a device with animated 3D information blended on top of it. Especially in combination with mobile devices Augmented Reality techniques can be excellently used by service technicians as a supportive medium. Due to their small sizes and the descriptive ability of VR-based techniques, mobile devices are an ideal medium for interactive on-site manuals.

As mentioned, AR is a method to combine the real environment with virtual information blended into the field of view of the user. Thereby, a step-by-step manual of reparation tasks can be created, which are presented in completely different ways, like pure virtual information or information calibrated over an image made by the technician on site. Of course a technician does not always need such a rich representation for each task. But often the best representation is to have an image of a machine part, which is augmented by an animated task to be performed. There are two ways of achieving the image, depending on whether a technician is satisfied by a preset image from a predefined viewpoint or prefers to make a snapshot from his own point of view.

Thus, several different kinds of task presentations are desired, which requires a software architecture that gives the ability to create arbitrary kinds of presentation in an easy and flexible way. Thereto, in [262] we presented an architecture that allows the creation of different kinds of task presentations. It is based on elementary modules, which can be linked together to create chains of building blocks that represent a special type of task. Ideas of applying virtual and especially augmented reality as a tool in the field of production, service or training evolved since the late nineties (cf. [262]), but most projects regarding AR maintenance depend on a pure AR approach to present work steps by augmenting the image with text or simple 3D models.

But to support a mobile and mostly inexperienced technician, multiple kinds of presentations can be suitable for fulfilling different steps of a whole workflow. Hence, especially in AR environments having a virtual character, which – similarly to a real instructor – explains the next steps while e.g. pointing to a certain part of the machine, can be the most appropriate type of presentation. Here, the X3D-based software architecture presented in chapter 7, can be utilized to quickly create such an individual task presentation chain by additionally providing means for authoring (see section 7.4) and to easily setup a time-line with different 3D animations and events.

6.5 Conclusions

In this chapter, we first have presented a declarative approach to camera placement for X3D, which allows people to specify what they really want to see where on the screen. The model is much more natural and allows framing characters and others objects very intuitively. Thus, a scene author does not need to navigate through the virtual world for finding the right viewpoints. By utilizing well-known guidelines from the film area, it allows to directly define what objects shall be framed, which makes standard camera setups easy. The camera model is useful for any kind of content, but it is especially of interest for dialog systems including avatars and complex objects, whose position the camera model uses to automatically calculate the final view matrix.

Besides camera control we discussed several other techniques for describing cinematic content including lighting and visual effects. The proposed camera model therefor includes special effects like motion blur and depth-of-field, which allows incorporating classical film effects into real-time scenes easily. The proposed X3D extensions not only allow creating camera moves that are rather close to traditional filming, but they also support automatic camera movements that are bound to interactive content even in fully dynamic scenes, which makes previously impossible dynamic settings now possible.

Moreover, the outlined cinematographic camera approach helps with framing subtle effects like blushing or crying that might otherwise be lost in the big picture. This includes real-time rendering techniques to present these effects in high quality, such as lighting and other effects like depth-of-field that can enhance the impression and are also useful for MR applications. Finally, to simplify application development, we have integrated the proposed techniques into the X3D standard and improved behavior control by wrapping the functionality with the PML layer, which will be explained in the next chapter.

Furthermore, in this chapter we discussed in how far the current X3D standard could be utilized for Augmented and Mixed Reality applications. Besides the general need for research in this academic field, this is especially of interest in order to allow integrating virtual characters as man-machine interface even in Mixed Reality applications with their special requirements in terms of hardware and realism. Depending on the type of application embodied agents can represent a much more efficient affordance within augmented environments than standard WIMP-like interaction techniques.

Currently X3D has some major limitations concerning the integration of external devices like cameras and tracking systems. However, since Instant Reality is utilized as development platform, previous research [24, 52] can be exploited. In addition, the virtual camera handling follows a simple pinhole camera model, which is not sufficient for AR applications. Finally the local lighting model in X3D does not support advanced rendering methods that include shadows, general multi-pass techniques and render state control.

To overcome these limitations we presented various enhancements to the present X3D ISO standard [151]. These comprise extensions to the viewpoint node on the one hand and nodes for advanced rendering techniques like irradiance mapping, shadows, and multi-pass methods on the other hand. Designing the new nodes and interfaces we focused on three main goals: the first is to be consistent with the ideas of the current specification, the second one is extensibility for future enhancements. And last but not least, our proposed node extensions allow application developers to create complex photo-realistic Mixed Reality environments easily.

Although the proposed shadow extensions are fast and stable and for many scenes work well in practice, the major drawback is that especially for large scale scenes they suffer from typical artifacts like aliasing. Another issue of the presented shadowing algorithms is the fact, that transparencies are not handled correctly in all cases and therefore mostly omitted during the shadow pass. This might be okay for a window pane but not for glass bricks or other semi transparent objects like sunglasses.

Albeit there exist approaches for special applications based on scene knowledge, there exists no general solution for arbitrary scenes yet. In combination with differential rendering this is an even bigger issue because parts of the real scene that are behind transparent objects are composed incorrectly. Another problem arises from the approach of using special shaders for the shadow calculations: user defined shader programs are ignored in the shadow pass, even those which transform vertices in world space or discard fragments, and therefore the shadows are calculated wrong in this case.

The idea to split the irradiance map calculations into a CPU and a GPU part, in which the final lighting reconstruction is done, is very fast and leads to convincing results. However, it has the disadvantage that it only works in combination with a specially designed shader and therefore the original material properties of the virtual objects, especially if they already contain shader programs, are mostly ignored here.

Nevertheless, the "piped" combination of current methods allows generating high quality images in real-time with real world lighting. New GPU-based techniques allow limiting the required processing power, thus making the system mobile and ready for further enhancements. We also described a new approach based on genetic algorithms to derive complex materials from the original scene radiance given by HDR images to minimize the

error in differential rendering for compositing real and virtual scene.

A great enhancement would also be to have a unified model for creating irradiance maps, because the currently used spherical harmonics for instance are unsuitable for high-frequency functions. Relating to actual reflection model parameters such as those of specular functions will then be much easier. In this regard, Wavelets seem to be promising, because the multi-resolution analysis allows capturing high frequencies with relatively few coefficients. Also, ambient occlusion as a placeholder for other surface functions is sufficient right now. However, due to their high computational costs, global illumination effects such as interreflections or caustics are currently not handled.

Moreover, for future work the proposed cinematographic camera model can be extended to allow for more than two characters or target objects, which can be achieved either by treating the camera pose calculation as an optimization problem or by implementing the hierarchical approach for groups of actors as discussed in [171]. In addition, including more abstract cinematographic concepts to encode camera moves [341] or certain shot types can be of interest. However, well-known placements like the so-called apex view or an over-the-shoulder shot already require semantic knowledge of the scene. Therefore, it would be helpful to connect the geometric data with appropriate semantic information (e.g. which part of the avatar denotes its shoulder) and means to cope with it.

7 Framework for Behavior Control

This chapter presents the conceptual framework for the proposed visualization component of multimodal dialog systems, which integrates all relevant functionalities and makes them available in a manageable way [183, 23, 156, 145, 147, 149, 160]. Therefor, the proposed framework combines important building blocks from the X3D-based execution layer with a declarative control layer, which allows for plausibly reacting virtual characters in dynamic environments. Whereas corresponding subproblems like camera control and emotion visualization have been discussed in the previous three chapters, in this chapter we mainly focus on the design of the control layer, how it is connected with the execution layer, and its integration into X3D and the Instant Reality framework on the one hand as well as into typical software architectures for dialog systems on the other hand.

7.1 System Integration

As mentioned, availability and efficiency is achieved by integrating the developed techniques into the X3D standard. In this regard, the proposed system and building blocks were integrated into the X3D-based Instant Reality framework [22, 135], which is a component-based Mixed Reality system developed and maintained at Fraunhofer IGD. This framework is utilized as visualization platform for a broader range of applications, since it is platform independent and already contains a lot of multimodal input and output interfaces such as head tracking [23], haptics [157], and multi-touch [152, 153].

Furthermore, it scales well from PDA [262] over web applications up to highly immersive multi-screen environments such as HEyeWall and CAVE [24]. Thereby it is possible to run the same application simply by using different engine configuration files e.g. as desktop or immersive VR. It utilizes the open ISO standard X3D [336] for defining scene description and runtime behavior, since representing the output as X3D means that it is easily distributable and sharable to others. Further, the framework utilizes OpenSG [232] for rendering, an open source high performance and high-quality scene-graph renderer. And due to its Computer Vision subsystem that provides marker-less tracking etc., Augmented and Mixed Reality applications are supported alike.

Within the system the behavior of the virtual world is defined by a scene-graph following and extending the concepts of X3D. Assets are thereby easily exchangeable, which is of great importance for content management. This description language only provides components for real-time graphics content, but not the applications in which they shall be embedded. The scene-graph [2] describes the geometric and graphical properties of the scene as well as its behavior. Each component in the system is instantiated as a node, which has specific input and output slots. Via connections between the slots of the nodes,

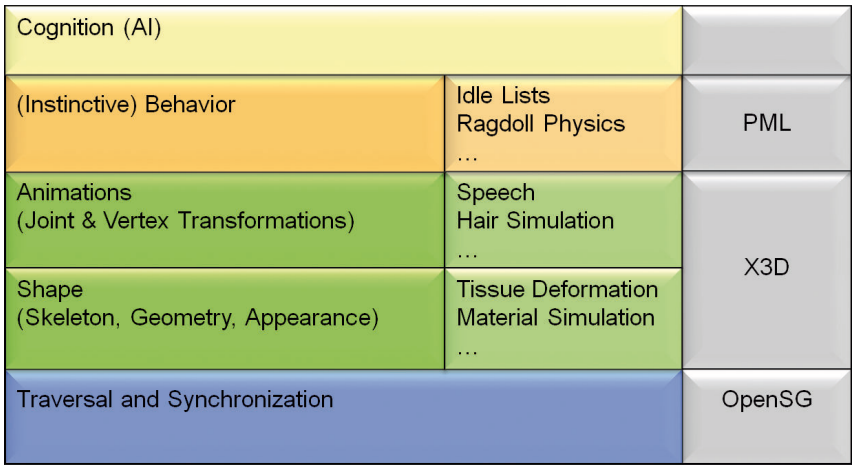

Figure 7.1: *Layers of complexity (right column shows basic technology used – top layer is not covered here): X3D/ H-Anim covers dark green parts (left column), but all orange and green parts should be covered, too, for simplifying application development.*

called routes, events are propagated, which lead to state changes both in the behavior graph and usually in the visible representation from render frame to render frame.

One major advantage of the component-based system architecture is that new functionality can be integrated easily by adding new node types and components [22]. Hence, for character animation the X3D H-Anim component [335] is used and further extended by the new *BehaviorController* component [135], which is presented in this chapter. Figure 7.5 shows the coarse idea on how this component is embedded into the X3D-based Instant Reality framework and how it interfaces with a multimodal dialog system.

7.2 The Control Layer

To provide a consistent and transparent interface, we have designed a centralized control component for animations and related actions [183, 156, 147, 149]. Similarly to the approaches outlined in Section 2.1.2, our declarative animation control layer (see Figures 1.4 and 7.1) is responsible for coordinating and synchronizing animations and events in time, e.g. "look sad and start crying". The control layer is not directly part of X3D [336], but builds on top of it by introducing the XML-based animation scripting and representation language PML.[1] This way, simplified application development is achieved by combining X3D for scene setup and PML for lower level behavior scripting beyond the tedious *Script* and ROUTE concept of X3D, as depicted in Figure 7.8.

7.2.1 Introducing an Animation Control Language

An important aspect concerning multimodal dialog systems is the flexible and powerful control of the behavior of virtual characters, including the ability to implement and author

[1]The full documentation of Player Markup Language (PML 2.1) can be found here: http://doc.instantreality.org/tutorial/controlling-behaviour/PMLSpecification.pdf

story-lines. In movies, virtual characters are used that are almost life-like, but which have completely scripted actions. In games, characters can be autonomous, but the interaction with them is unnatural and limited to pre-programmed commands and behaviors.

As can be seen in Figures 7.5 and 7.7 showing the internal and external view of the framework, a predominant aspect for the design of the visualization component is the requirement that it is able to expose its functionalities to the higher-level modules in such a way that specificities are abstracted away and that they can control character animations and other behavior without the need for dealing with low-level graphics concepts.

If an H-Anim figure [335] within X3D shall be modeled with more advanced behavior like a set of skills and typical actions, this currently needs to be done externally by using the SAI [337] with a Java interface, what mostly not only means slow value updates and multiple data storage, but often also to reinvent the wheel. By providing an additional layer on top of H-Anim for scripting and synchronizing animations and other motions based on our proposed animation control mechanism, a lot of complexity can be avoided, and the application developer can concentrate on the story and scene development itself.

Therefore a higher level of abstraction is needed, which will be described by an additional control language. It provides an abstract layer to the graphics environment and its usage is also suitable for non-graphics experts. It therefore allows controlling character behavior and emotional states in any desired temporal order not only by researchers in other areas like artificial intelligence or story-telling but also by designers.

Recently, Heloir and Kipp [123] outlined the need for an additional animation layer in the context of dialog systems, which is a thin wrapper around the animation engine and situated below higher-level behavior control layers. Likewise, we have integrated a scriptable animation controller component into our framework to rapidly describe life-like characters. But in contrast to their control language "EMBRScript", ours is based on XML and does not deal with engine-specific key poses, in order to ease development and module integration, though it is low-level enough to be directly understood by our animation system. Thus, in this section a domain specific language is introduced for controlling virtual environments with special regards to animation, which is comparable to the additional programming languages needed for X3D *Script* and *Shader* nodes

7.2.2 Player Markup Language

The descriptive XML-based high level markup language PML (Player Markup Language) on the one hand is based on concepts taken from RRL [247] such as the notion of scene descriptions and scene acts stemming from theater terminology as well as gesture assignment, which at the end has to be transformed into a player-specific format. Both focus on the specification of communicative behavior of virtual characters in multi-party dialogs as well as on a primarily system-internal use. In this regard RRL is a formal framework for representing the information that is exchanged at the interfaces between the various system modules necessary to represent the behavior of conversational agents.

However, RRL does not support standard media types such as videos or GUI elements, which makes it unsuitable for highly interactive multimodal scenarios. On the other hand PML has capabilities for declarative 3D animations via SMIL-like animation elements

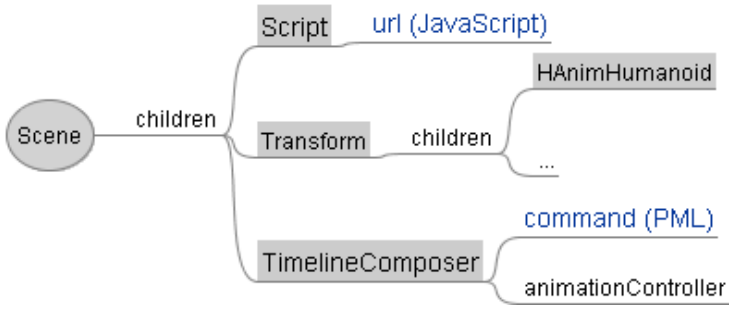

Figure 7.2: *Usage of the animation scripting language PML within X3D.*

[323]. The Player Markup Language was developed corporately with researchers from the area of artificial intelligence [100, 183, 181, 156, 147, 149] and results from the Virtual Human project [321] and, amongst other smaller industrial projects, is used and further extended within the ANSWER project [6].

Originally PML was designed as a representation and interface language between several components like for instance a dialog and an affect engine [204, 181] on the one hand, which are AI-driven, and a graphics engine on the other hand [156]. Therefore, PML is designed to be independent of the implementation of a 3D engine and virtual environment. Besides, when animating and visualizing virtual characters one also has to think about interoperability aspects. Especially in web environments, it should be possible to specify the properties and behaviors of characters and objects in a virtual environment independently from their realization in a concrete setting, whilst still being able to provide detailed information like the required animation parameters and exact timing information.

PML hence can either be used as descriptive interface markup language between a graphics engine and some higher level behavior and dialog generation engines (see Figure 7.3), or for directly scripting animations [147]. It can thus be considered being a control language for scripting the proposed animation control component comparable to JavaScript for scripting the *Script* node (with which it can be combined), and thereby extends the current rather basic X3D concepts concerning scripting and synchronizing animations within an X3D environment (as shown in Figure 7.2). Furthermore, due to its generic design it is able to synchronize conventional X3D scripts in time. Though PML focuses on the specification of behaviors of virtual characters it also contains elements to specify and coordinate the presentation of other scene elements over time.

7.2.2.1 Defining a Stage with PML

Because PML is a specification language for controlling virtual environments with special regards to character animation and user interaction, it defines a format for sending appropriate commands. Additionally, it defines a message format, which can be sent to the animation control component or received from it for enabling interactions with the scene via `<message>` and `<query>` scripts. At the beginning of a new scene all objects and characters are defined by a `<definitions>` script.

There exist three main types of definitions: repository definitions, character definitions, and object definitions. Repository definitions specify the path where the resources for the

various scene elements are located. Character definitions specify the acoustic parameters of the synthetic voice, the available animations including their default durations, the phoneme to viseme mapping to be used, etc. Each of the characters animation is labeled with an 'id', which is unique within the character scope. Likewise object definitions are used to specify cameras, user interface elements, and various media types that will be used in the scenario, whereas the 3D objects is required to support these features. It is the responsibility of the PML author, that only supported objects will be assigned. A short example that defines a list of idle animations is shown next.

```
<definitions id="iListDef">
  <character id="Valerie" src="Valerie.wrl" root="Trafo">
    <multiPoses id="a" src="a.wrl" dur="2350"/>
    <multiPoses id="b" src="b.wrl" dur="2533"/>
    <idlePoses id="iP" random="true">
      <multiPoses refId="a" dur="2350"/>
      <multiPoses refId="b" dur="2533"/>
    </idlePoses>
  </character>
</definitions>
```

Idle behavior, like breathing, blinking with the eyes, moving slightly around, and similar noise-like motions, is displayed in between when no specific animations are active, because it looks rather unnatural, if a virtual character stands absolutely still [75]. Even better than simply repeating a breathing or blinking animation is to randomly choose between some more interesting actions like looking around or wiggling the foot due to a certain unpredictability of life-like behavior in general [260]. Here, the `<idlePoses>` element defines a list of animations, which will be played when a character is idling. With the 'random' attribute the playing order of the animations can be set. So the list can be played randomly or in the predefined order for the character specified with its surrounding tag. Whereas the 'src' attribute of the `<character>` element references the file that contains the character's basic geometry, the optional 'root' attribute denotes the name of the node, which defines the character.

Each PML element has a unique 'id' that relates to one virtual object and by which it can be referenced via the 'refId' attribute in other elements. Identifiers can be generated at runtime or assigned statically. According to their usage, they must be unique over the entire system or just in their current context. At the beginning of an application, the definitions scripts have to be loaded. Within an `<actions>` script, it is then possible to refer to the previously defined animations of a character. The script thereby schedules the movements of the virtual character. Because each PML document has exactly one of the four mentioned top-level elements, its grammar in BNF is defined as follows:

```
PML2DOC ::= DEFINITIONS | ACTIONS | MESSAGE | QUERY
```

All characters, 3D objects and animations, which will be used in specific actions, must be defined in advance. Scene definitions are used to tell the player, which scene elements

will be used in future scripts, and how the associated data can be loaded. Every scene element that shall be used first has to be registered and labeled. Definitions scripts can be sent to the player at any time. They will be executed by the player immediately after receiving. All required data will then be loaded into memory, and all scene elements will be instantiated and registered immediately.

With the fourth definitions type, the <undefine> element, already loaded scene objects can be removed from memory. The scene object to remove is specified by the 'refID' attribute. Hence, the grammar that defines a PML definitions script, as given in BNF (Backus-Naur Form), looks like follows:

```
DEFINITIONS ::= <definitions id="ID">
                  REPOSITORY*
                  UNDEFINE*
                  (CHARACTER | OBJECT)*
                </definitions>
```

7.2.2.2 Handling Animations and Events

An actions script consists of a number of activities and assigns them within a temporal context. Here, only such scene elements can be used that were defined previously in a definitions script. In the course of the story all runtime dependent actions like character animations are described by so-called <actions> scripts, whereas the temporal order is given by a special scheduling block including sequential and parallel elements. Hence, a PML script describes a scene act, and all PML actions can run successively or simultaneously. An actions script is executed only once, and after execution it will be deleted from memory. The grammar of an actions script basically is defined as follows:

```
ACTIONS ::= <actions id="ID" start="BOOL">
              (CHARACTERACTION | OBJECTACTION)*
              SCHEDULE 0,1
            </actions>
```

Actions are used to specify the appearance and behavior of all characters and objects in the environment. Some actions like 'show', 'hide', 'transform', or 'startIdleList' can be applied to both, characters and objects, while others are only available for specific scene elements. Every action is labeled with a unique id attribute. Below a short example script is shown, in which the previously defined idle list is started. Examples of actions that are only available for virtual characters are 'speak' for verbal output, and 'complexion' for the change in skin color, like blushing and pallor.

```
<actions id="iListStart" start="true">
  <character refId="Valerie">
    <startIdleList id="iL" refId="iP" />
  </character>
  <schedule>
    <action refId="iL" begin="0" dur="0"/>
  </schedule>
</actions>
```

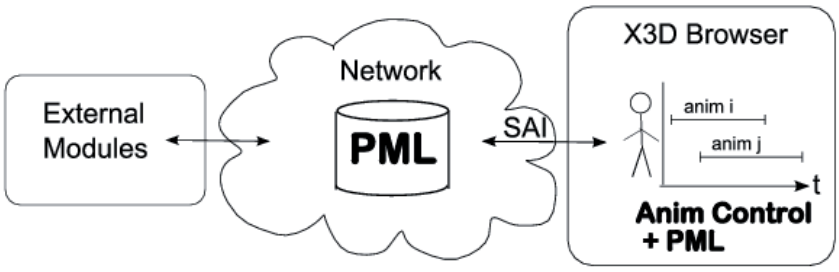

Figure 7.3: *An essential additional use-case: Utilizing SAI to send PML chunks during runtime to the animation controller.*

Using `startIdleList` and `stopIdleList` can activate or deactivate `idlePoses`, which can either be a `CHARACTERACTION` or an `OBJECTACTION`. By specifying the attributes *minDelay* and *maxDelay* a time interval can be defined, which is quite convenient for e.g. simulating blinking. In this case, the player is waiting n milliseconds before the next idle gesture will be played, where n is a randomly selected value from the specified interval. The grammar is defined as follows:

```
IDLELIST ::= <startIdleList id="ID" refId="ID" [concurrent="BOOL"]
              [minDelay="UINT"] [maxDelay="UINT"] /> |
             <stopIdleList id="ID" refId="ID" />
```

The task of the schedule block, which can be created by any higher level PML generating module or authoring tool, is to tell the visualization engine, when which action shall be executed. As can be seen in the example, the `<schedule>` element describes the temporal sequence of actions within a schedule block. Similar to SMIL [323], actions that should be executed sequentially, are embedded in a `seq` element, whereas parallel actions are embedded in a `par` element.

The `<animate>` tags of a PML actions script can either refer to predefined animations, which are referenced by their name, or to simulated animations, e.g. via inverse kinematics. Different kinds of animations like morph targets and displacers for facial animation (given as `<singlePose>` child element), or key-frame animations (`<multiPoses>`) and simulated animations (`<implicitPose>`) for gestures and postures are distinguished, because every animation type must be handled differently and has a varying set of attributes.

An example of a rather unusual animation which can be handled quite easily this way, too, is the change of the face complexion. Usually only the changes in geometry by means of displacers or morph targets are addressed in computer graphics. This is a well known problem, and the classification usually is based on the Facial Action Coding System [77], which identifies certain Action Units for morphing the face geometry. But with the help of modern graphics hardware the more subtle changes concerning face coloring can also be covered via animated skin textures or shader programs (see Figure 5.12, bottom). Hence, facial color changes are determined with the `<complexion>` element, which, like `<animate>`, is another possible `CHARACTERACTION`.

```
CACT ::= <complexion id="ID" refId="ID" dur="UINT" intensity="FLOAT" />
```

A PML message is used to control the execution of actions and to exchange information between modules. There are three different types of messages: commands, states, and facts. Commands can be used to trigger the execution of actions; states are used to inform other modules about the execution state, e.g. started, failed, or finished, what is important for later synchronization; and facts, which are represented by attribute-value pairs, can be used to inform about user actions (for example user has chosen answer c). Finally, a query can be used for retrieving scene information.

For the corresponding commands and status messages the following rules apply. If the player receives either a definitions or an actions script, it send the status *fetched*. In a message or query script, nothing will be transmitted. Actions scripts must be started explicitly, either with the help of a start command in a message or by directly using the start attribute. All other scripts start automatically after they were received. As soon as the first action in a actions script runs, the player sends back the status *started*.

If an actions script will be terminated prematurely due to the stop command, the player sends the status *stopped*. If the player fails to execute an action, it sends the status *failed*, referencing the id of the failed script and optionally an error message. If a definitions or actions script is fully processed, the player sends the *finished* status. In a message or query script no status will be returned. To get all modules back to their initial state, a *reset* message will be send. This message causes the player to discard all definitions and abort all running actions.

By introducing a more abstract mechanism to define and synchronize different kinds of animations without having to take care about correct routing, timing etc., it is also much easier to create digital stories with embodied conversational agents in X3D. Such a story can be described with PML by putting together story-lines, i.e. short scene acts, in an easy and intuitive way through PML scripts that define when and what a character or object in the scene is doing. By combining this with other script or sensor nodes that define when and how the user can interact with the virtual environment there can also be added some non-linearity and possibilities for user interaction in order to create an interesting story graph. Figure 7.3 shows a possible system setup. As can be seen, PML can either be used for scripting and synchronizing within the X3D browser (cp. Figure 7.2), or for handling the communication with modules that do not want to bother with problems concerning low level kinematics.

7.2.3 High-level Runtime Control

7.2.3.1 Controlling Emotions

As mentioned, in the context of AI a layered model of affect for enhancing simulated dialogs was introduced by [91]. Three types of affect based on different temporal characteristics are distinguished: emotions, to which e.g. facial expressions belong and which are short-term affects; moods; and finally personality, specifying the general behavior. After a character's personality profile is set, his affective state is updated in real-time, and is used to change gestures and control facial expressions. Amongst other things the Player Markup Language (PML) was developed in this context.

Here, PML is used as a descriptive interface between a dialog engine and a renderer by defining a format for sending commands and exchanging messages. All runtime dependent actions are described by actions scripts, also containing the schedule. The communication with our X3D-based VR system is handled by a special scene graph node which forwards the PML commands to another special controller node, referencing the corresponding character. This way, no semantic model is needed for the renderer and also unusual animations like temporal changes of the complexion can be handled easily.

Obviously the signs of strong emotionality need to be controllable the same way as other character motions for being consistent with gestures and facial expressions. According to the concepts of X3D that only provides lightweight components for 3D graphics content, no further semantics or "intelligence" is connected to the scene-graph nodes. As mentioned, there is no notion of a special object *Eye* – it is simply a mesh with a certain appearance.

Thus, for being used in the control layer, first the droplet flow simulation as well as the skin shader together with the intensity need to be wrapped by "AnimationContainer" nodes and linked with the corresponding PML definitions script via the name–id mapping. Then, everything can be used in a unified manner at a higher level via PML actions. The following simple PML actions script, which combines typical facial animations like blinking with blushing and crying, shows how such an animation can be controlled [145].

```
<actions id="act2" start="true">
  <character refId="Susan">
    <startIdleList id="idle1" refId="Blink" />
    <complexion id="L1" refId="redHead" />
    <complexion id="L2" refId="tears" />
  </character>
  <schedule>
    <seq>
      <action refId="idle1" begin="0" dur="0"/>
      <par>
        <action refId="L1" begin="0" dur="4000"/>
        <action refId="L2" begin="0" dur="4000"/>
      </par>
    </seq>
  </schedule>
</actions>
```

7.2.3.2 Camera Control

Because, as mentioned, an additional layer between animation engine and high-level control is necessary [123], we have extended the scriptable animation controller component to realize both, character and scene/ camera control. Designed to also embrace cinematographic concepts, the declarative PML interface not only provides a unified and flexible control of virtual characters but also for the cinematographic camera node [149]. Thus, it can be easily parameterized from within any higher level engine.

For dialog scenes with strong emotional focus the latter is essential, because cinematography, as the art of motion picture making, is concerned with communicating the emotional

state of an actor or establishing relationships between actors mainly via camera work. Therefore, we have integrated support for controlling virtual cameras into PML. The proposed <camera> element is likewise defined as a special scene object with certain functionalities like <zoom>, <startFollowing> and <frameTarget>. The latter element allows setting the shot size (e.g. "closeup"), the offset angles and the minimum and maximum bounding box positions of an object or character in normalized screen coordinates following the rule of thirds. The screenshot shown in Figure 6.2 (v) on page 167 for example was obtained for the camera part by running the actions script depicted below.

```
<actions id='actCam' start='true'>
  <object refId='CineCam'>
    <frameTarget id='e' refId='cam' shotSize='extremeCloseup'>
      <minScreenPos> 0 0 </minScreenPos>
      <maxScreenPos> 1 1 </maxScreenPos>
    </frameTarget>
  </object>
  <schedule>
    <action refId='e' begin='0' dur='0' />
  </schedule>
</actions>
```

In addition, to enable high-level control also of visual effects as discusses in Section 6.2.4, we extended PML by means of the <activateEffect> entity that allows a higher level engine to activate a certain visual effect for the currently bound camera, to which it refers with its 'refId' attribute [160]. The type of effect to be activated is specified via the correspondent 'type' attribute (e.g. "depthOfField", "blur" or "sketch").

7.3 Framework

7.3.1 Analysis and Layer Design

As already outlined in section 4.2 (p. 102), we follow a layered approach for simplifying scene setup and behavior control in X3D by splitting up the complexity into different levels of abstraction [147, 149], which is shown in Figure 7.1. Both upper levels make up the control layer, whereas the lower X3D-based levels constitute the execution layer (shown in green). In addition, here we further distinguish between consciously controlled behavior and "unconsciously" happening phenomena, since these are conceptually different.

The upper levels, namely the control layer, which in Figure 7.1 is shown in amber and beige respectively, deal with more instinctive behavior as well as cognition. X3D provides some support for both lower layers, but there is no support for the upper ones. Whereas cognition, which deals with decision making and emotion simulation and controls motor functions, belongs to the field of artificial intelligence and is beyond the scope of this thesis, autonomous or scripted behavior in the sense of basic skills should at least partially be handled here, although for X3D the focus clearly lies on the graphical representation.

An example of such instinctive behavior is the more or less unconscious idle behavior like breathing or blinking with the eyes mentioned in section 7.2.2.1. Other autonomous behavior can e.g. be represented by the famous boids model [260] or the more advanced steering behaviors described in [261], though obviously it is not well suited for conversational behaviors. Either way, the X3D runtime needs knowledge of these animations as well, because if suddenly an intentional action shall be executed, the idle animations mentioned in section 7.2.2.1 cannot simply be stopped but have to be blended over into the new animations to avoid artifacts (see Figure 4.4).

The X3D-based execution layer thereby only deals with geometry, appearance, and simple animations and dynamics (as shown in green in Figure 7.1 on page 204). In these layers consciously controlled behavior resulting from motor control, like speech, mimics and gestures, is handled, which was described in the first part of chapter 4.

Additionally, this layer is also responsible for treating phenomena that cannot be controlled deliberately. As was already outlined in section 4.2 on page 102, they can be further divided up into (psycho-) physiological processes and subconscious behavior that for consistency needs to be triggered externally like crying or blushing on the one hand, as well as dependent effects that simply happen such as hair motions on the other hand.

Thus, besides speech, body animations and the like, there still remain some aspects, which cannot be scripted at all. These namely comprise of adjoint effects that usually directly result from an animation or similar actions: if e.g. a character with long hair shakes his head, not only the hair has to follow obeying the laws of physics [330], but also the shadows must change correspondingly. Whereas these fields of research are mostly not unique to characters, they need to be considered nonetheless.

The final rendering, including its distribution in cluster or multi-screen environments, is handled by the lowest, OpenSG-based layer [232], which is shown in blue in Figure 7.1. This rendering layer is not directly exposed to the user or other modules [149].

Hence, the structure of the proposed framework conceptually can be described as follows:

Control Layer Coordination and synchronization of actions and events in time for modeling the behavior (represented by the two topmost rows in Figure 7.1)

- Abstraction for behavior description via scripting and interface language PML
- Simplified integration into module pipelines by using declarative approach

Execution Layer X3D-based building blocks to describe appearance and dynamics of virtual environment and characters (marked in green in the middle of Figure 7.1)

- "Consciously" controlled behavior
  - Mimics, speech
  - Postures, gestures
- "Unconsciously" happening phenomena
  - Tears, blushing
  - Hair movements
- Virtual environment
  - Camera, lighting
  - ...

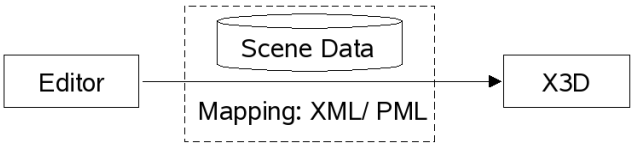

Figure 7.4: *Exemplary system communication and data mapping via a Gesticon.*

In order to simplify character animation and related events in X3D, in section 7.2 we have presented the interface and control language PML for specifying and synchronizing animations and similar actions at a higher level. Because this requires to have the accordant features on the lower X3D-based content levels, we furthermore proposed a set of scene-graph nodes for realizing these demands in chapters 4, 5, and 6.

To bridge the gap between these layers we also propose some nodes for converting the scripted schedules and for controlling animations, which are capable to mix an arbitrary number of interpolation-based animations. Though in the course of this work, with respect to character animations, we focus on gestures and mimics, as these are the most relevant for multimodal dialog systems, the presented framework is kept extensible to new concepts of on-line motion generation, e.g. for locomotion, too (cp. [183, 156, 145, 147, 149]).

Thus, in order to handle PML commands, first of all a generic scheduling and controlling element is needed. Therefore, we introduce an additional animation and behavior control component to X3D, whereas the "TimelineComposer" node acts as PML interface and processor. Section 7.3.4 provides more information and some examples. Though X3D provides several types of scripting interfaces, thereby allowing for dynamic and interactive virtual worlds, this imperative approach of controlling object behavior with an API is orthogonal to the declarative and document-based design of X3D.

This new *BehaviorController* component [135] – depicted by the upper green parts in Figure 7.10 – additionally provides means for combining animations with the help of the "AnimationController" node, which was briefly referred to in section 4.3.1 (see also Figures 7.5, 7.8, and 7.9). Since in this regard animations and the like are exposed as service for other applications, requirements like blending and cross-fading can be derived from higher levels and must be provided in a manageable manner.

7.3.2 Asset Management

The aforementioned additional *BehaviorController* component contains nodes like the *TimelineComposer*, which can be used for scripting and synchronizing behavior in X3D scenes by means of the interface and description language PML (cp. section 7.3.4). By introducing PML [147, 149], simplified development of X3D applications can be achieved, but what is still missing, is to include the process of asset creation and setup, which nowadays still affords time and manpower as well as expensive tools and experience. Hence, in this section we will outline how to alleviate these issues [160] towards a simplified and comprehensible content pipeline based on X3D.

First, the 3D assets including all character animations need to be created, which is usually done using a modeling tool like Maya or 3ds Max, and exported into X3D format.

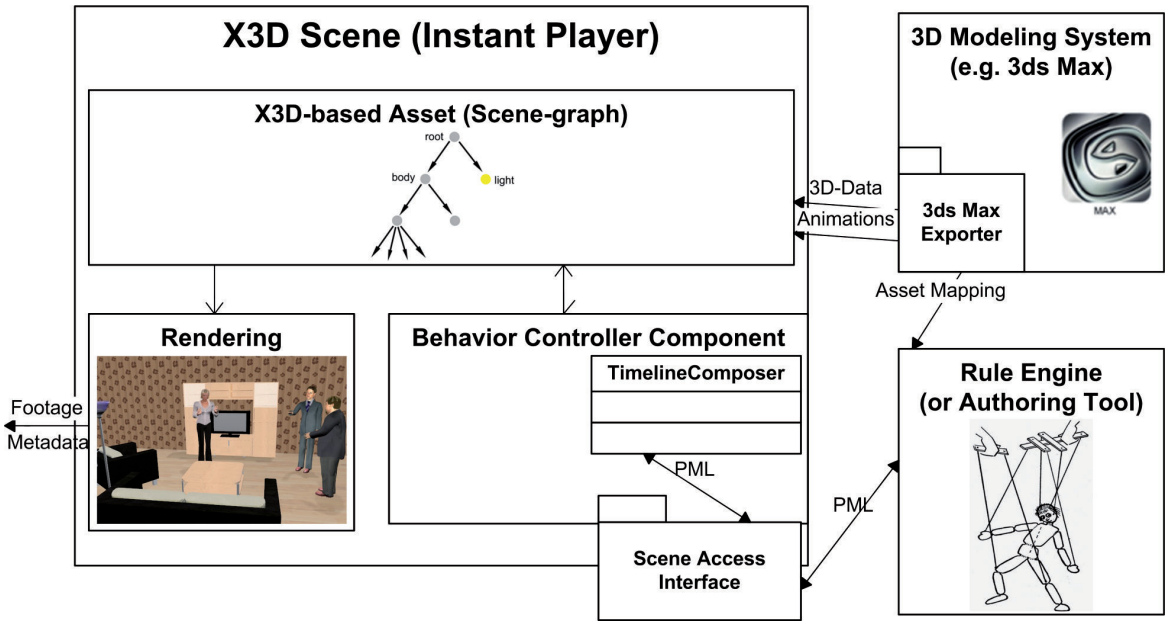

Figure 7.5: *Internal view with framework architecture and interfaces to external components.*

Because the controller component does not care about behavior creation, almost all behavior/ animations that shall be used, first needs to be defined and specified within X3D. Therefore, we have extended our 3ds Max exporter to be able to directly export animations as *AnimationContainer* nodes (its interface is shown in Figure 4.2, p. 100).

Furthermore, because higher level modules often deliver elliptical "stories" based on behavior classes and still lacking concrete scene information, this data then has to be enriched with scheduling information and translated into a more concrete representation by mapping the higher level concepts given e.g. in DirectionML (cp. Figure 7.7) to a less abstract description of scene and animation data (e.g. load and play animation "sitDown.x3d") in order to be executed by a generically designed X3D browser and to additionally achieve portability to other environments like a game engine. This process is depicted in Figure 7.4, and is achieved with the help of a so-called "Gesticon" as described in [194]. A very simple example of a Gesticon defining a character and its surrounding environment (as was used for instantiating the scene shown in Figure 6.6) is shown next.

```
<characters name="Bandit" description="A bandit character"
            file3D="bandit/Bandit.wrl" rootNode="Bandit_TRAFO" >
  <animations type="shrug" animType="keyframe" name="Bandit_shrug"
              dur="2133"   file3D="bandit/shrug_TAC.wrl" />
  ...
  <!-- optionally properties and bodyParts -->
</characters>

<stages name="Chapel" description="A chapel's interior"
        file3D="stages/mainChapel.wrl" rootNode="chapel_TRAFO">
  ...
</stages>
```

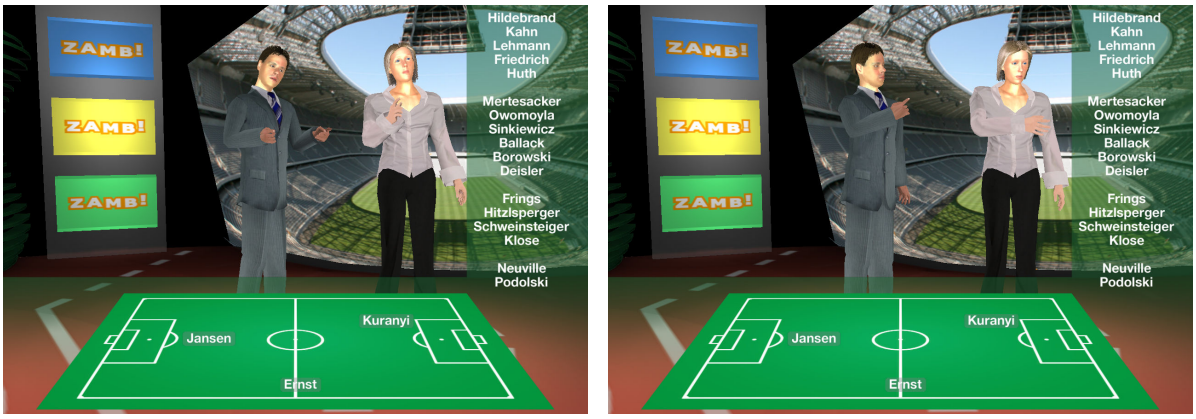

Figure 7.6: *The Virtual Human [321] Demonstrator ZAMB (compare Figure 1.3) was shown at the INTETAIN '05 conference and on CeBIT 2006.*

Thereto, a preferably XML-based dictionary containing all assets including their semantic meaning has to be generated in advance, i.e. it has to be declared, which file/ node contains what object, gesture, etc. This is necessary to enable the concrete mapping from animation containers (as explained in section 7.3.4) to certain PML commands, as visualized in the upper right part of Figure 7.5. As mentioned, a rule engine (cp. Figures 7.7 and 7.5) here provides the translator necessary to transform user input in the higher level modules into the formalism needed for 3D visualization purposes. As all higher level descriptions are given in XML they can easily be transformed and enriched to more concrete descriptions with standard tools and methods such as XSLT.[2]

7.3.3 System Communication and Module Integration

In order to explain a typical system setup in the area of multimodal dialog systems, exemplarily a diagram of the component-based Virtual Human [321] architecture (i.e. the external view) is shown in Figure 7.7. A more in-depth description of this project can be found e.g. in [100] and Figure 7.6 shows some results. The framework follows the typical pipeline model and is based on a distributed web service architecture model, which is common in the area of AI and allows most of the components to be sufficiently decoupled from others in order to enable both, interdependent and independent functioning.

The content layer of the Virtual Human platform consists of a domain model providing geometry, story and character models, pedagogic models, media, etc. Furthermore, special editors can be used to create application scenarios and stories. The output of the content layer is a story-board. The narration engine consists of a story engine controlling a scene engine and an improvisation module [136]. The conceptual semantic representation of the dialog elements introduced in the narration engine is covered by the ontology. Hence the narration module or authoring tool, which enables the user to represent a scene, will generate dialog instances that are mapped to ontological concepts.

The outputs of the story engine are directions coded in DirectionML (top left of Figure 7.7) for the Dialog Engine [204, 181]. While the the dialog is generated, this information

[2]Extensible Stylesheet Language Transformations, http://www.w3.org/TR/xslt20/

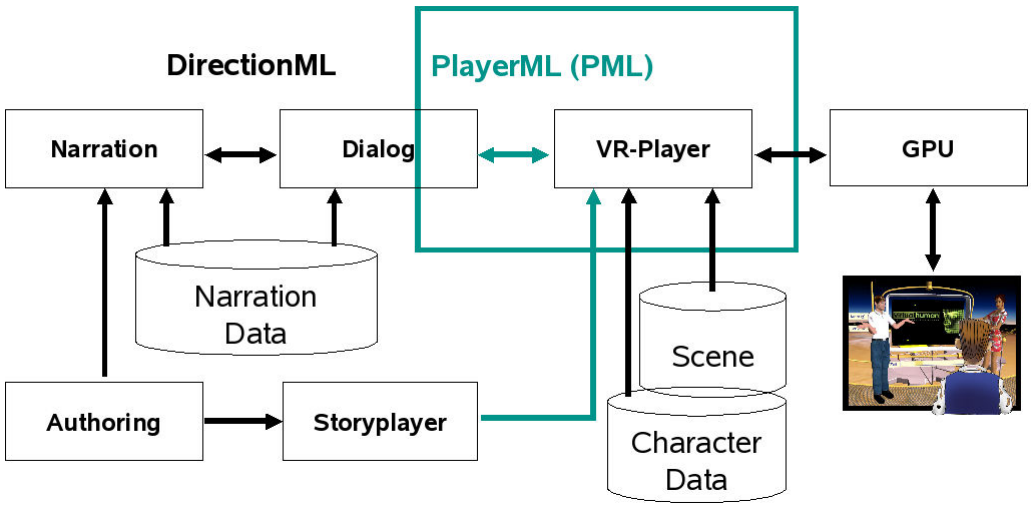

Figure 7.7: *External view: the Virtual Human system architecture and module pipeline.*

is passed over to the Dialog Engine, which performs semantic analysis of the story, applies reasoning based on the underlying ontology, and converts the dialog into a more enhanced discourse model that is enriched with deictic and other conversational gestures. Here the – on linguistic grounds – elliptical input, which uses relative timing and has behavior classes like "head nod", is transformed into its fully resolved form, which is still high-level compared to the inputs of a typical graphics engine, but is complete in the sense that all required information is provided explicitly rather than implicitly.

Therefor, the narrative input, as delivered by the Dialog Engine and given in XML format, then has to be enriched with scheduling information and translated into a more concrete representation by mapping the higher-level dialog acts to a less abstract description of scene and animation data given as PML scripts [204] with the help of the aforementioned Gesticon (compare Section 7.3.2). Here, dialogs among virtual characters are generated during run-time and sent to the player component as PML scripts.

Finally, the visualization module transforms the results into a rendered and interactive 3D animation and sends corresponding data to the visualization platform. Possible peculiarities of such platforms range from simple monitors or web-based application scenarios up to high-level rendering applications to be shown on a Powerwall.

To prove that concept and methods are equally suitable for different domains, we designed and realized a similar architecture for creating animated 3D pre-visualizations of films within the ANSWER project [6], where the scenario is generated based on a symbolic language for describing actors and camera work. The corresponding framework architecture already was coarsely outlined in section 6.2. In order to additionally show that PML is designed generic and player independent enough to be useful in other non-X3D-based environments, here also a game engine's output is controlled via PML.

In short, in Virtual Human [321], PML is used as descriptive interface markup language between a dialog creating environment (dialog engine) and the VR player and synchronizes all steps of dialog generation. In the first step of the dialog generation (i.e. in the first component), a very generic representation of the dialog is generated. In the next step

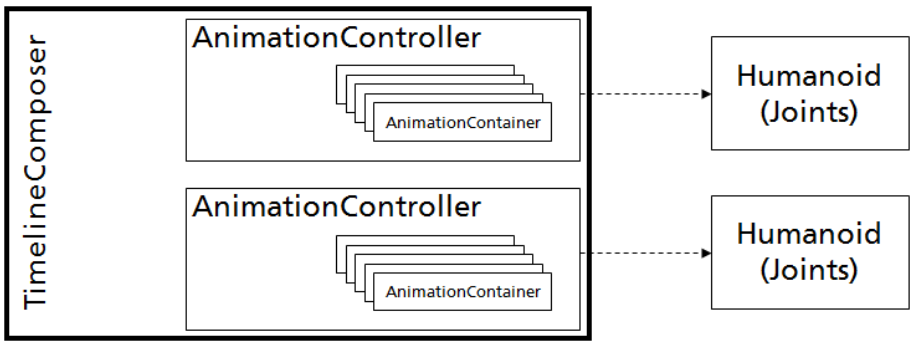

Figure 7.8: *Character data organization for PML: the TimelineComposer node acts as the PML interpreter, and references an AnimationController node for each humanoid, which controls and synchronizes its speech output, animations etc. (provided by AnimationContainer nodes) and also supports idle lists and animation blending.*

additional information (e.g. emotions, timing for lip-synchronization) augments the dialog-skeleton. In the third step, the dialog is enhanced with appropriate animations like mimics and gestures. The result is a script with a full definition of the actions and their temporal order. In a complex scenario, such an actions script encodes durations of 5 to 20 seconds, meaning that the player receives many scripts in a short time. Nevertheless, such actions scripts can also encode a much longer duration [183].

7.3.4 Scheduling and Controlling Animations

7.3.4.1 Connecting the Layers

After having explained the low-level extensions in the previous three chapters as well as the high-level language PML in section 7.2, the question remains, how this advanced animation control approach can be used in concrete settings. Thus, a generic scheduling and controlling element is needed, too [156, 147, 149]. Therefore, we added some additional nodes, whose X3D interfaces are discussed next. In addition, Figure 7.10 shows an overview of the proposed system architecture and how it is embedded into an X3D graph.

Here, the proposed *TimelineComposer* node is responsible for all scheduling and also deals as the PML interface and processor (cf. Figure 7.8). When all content is created and setup, a rule engine or any other higher level engines or authoring tools can send PML commands via the network layer/ SAI [337] to the *TimelineComposer* (see Figure 7.3). Figure 7.5 outlines how the controller component is embedded into whole system. Alternatively, PML can also be used for declarative animation scripting directly from within an X3D world as visualized in Figure 7.2.

```
TimelineComposer : X3DNode {
  SFBool   [in,out] enabled TRUE
  MFString [in,out] command []
  SFString [out]    message
  MFNode   [in,out] animationController []
}
```

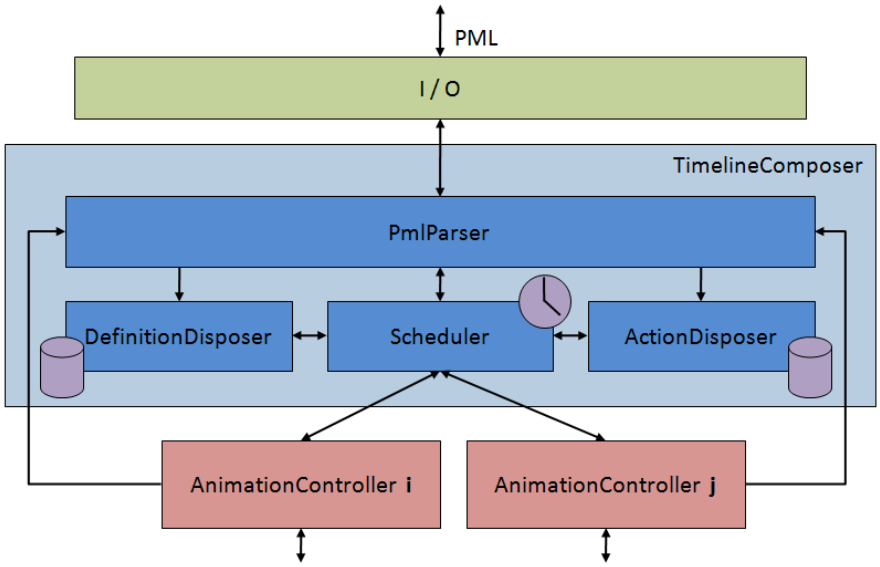

Figure 7.9: *Architecture and interfaces of proposed control component (cp. Figure 7.8).*

Starting and stopping of animations and other events, which are given as *AnimationContainer* nodes, is accomplished by setting the 'command' string of the *TimelineComposer* node with a valid PML file or string for defining the desired temporal order (see Figure 7.9). This is similar in spirit to the 'url' field of a *Shader* node, which only is useful when having defined a valid GLSL or Cg shader program, or the 'url' field of a *Script* node, which only is useful when having defined some Java or JavaScript code.

Whereas the 'command' field contains an incoming PML script, the 'message' eventOut sends an outgoing PML message string. This way, the *TimelineComposer* node handles all communication with the system and forwards PML commands to its parser as can be seen in Figure 7.9. During parsing, the scheduling block is sequenced and single action and definition chunks are created and transfered to the appropriate components. When having received a start message, the internal scheduler dispatches the action chunks to the *AnimationController* node of the corresponding character or object. The scheduler is also responsible for controlling visemes and audio output as well as idle gestures.

The MFNode field 'animationController' holds references to the *AnimationController* nodes of all objects, which shall be animated. Whenever an actions script shall be executed, the *TimelineComposer* triggers all *AnimationController* nodes, which in turn access the respective data of their referenced animation container nodes (the *InstantAnimationContainer* for referring to transitions, which are state changes like toggling visibility, and the *TimedAnimationContainer* for storing all time based animations like key-frame animations and inverse kinematics) for processing this request.

7.3.4.2 The Animation Controller

Temporal control including dispatching all active actions is done by the scheduler as visualized in Figure 7.9. The *AnimationController* node controls the set of animations

connected with a virtual character or any other object [156, 147, 149]. Because a larger application can lead to an arbitrary number of postures and gestures or respectively animations, the main job of the *AnimationController* is to blend and cross-fade all kinds of animations (cf. section 4.3.1), which are hold in its *AnimationContainer* child nodes (Figure 7.8). This is due to the requirement, that for correct blending, cross-fading, and generally updating the actions of an object at a single time step, the controlling unit needs knowledge of all animations, a task that often cannot be handled with the simple scripting and routing mechanisms of X3D.

```
AnimationController : X3DAnimationBase {
  SFString []        name               ""
  MFNode   [in, out] animationContainer []
  MFNode   [in, out] ikTargets          []
  SFFloat  [in, out] fadingInterval     0.2
  SFFloat  [in, out] fadingRotTol       0.7
  SFFloat  [in, out] fadingPosTol       3.0
}
```

The 'name' field contains the name of the object to be controlled, which is relevant for the later mapping to PML scripts: X3D 'name' and PML 'id' must correspond for enabling the mapping. The 'animationContainer' field contains references to all animations as defined by the animation container nodes. With the help of the 'ikTargets' field possible IK targets can be given, what is needed for parameterizing inverse kinematics animations like "look at A" or "point at B" via an `<implicitPose>`. If dereferencing is done on the PML layer as outlined in section 7.3.2, this multi-field is internally set during runtime.

Although blending avoids jumps in transitions, it can cause undesirable side effects like foot sliding. Therefore the fields 'fadingInterval', 'fadingRotTol', and 'fadingPosTol' can be used to specify the time interval and distances where blending should occur (refer to section 4.3.1 for implementation details). The default values were empirically determined and led in most cases to the best results.

The *AnimationController* and the abstract *X3DAnimationContainer* presented in section 4.3.1 (p. 104) both inherit from *X3DAnimationBase*, an abstract base node that only defines an SFString 'name' field. The *TimedAnimationContainer* node additionally contains a set of interpolators of an animation (in the 'interpolators' MFNode field – cp. Figure 4.5 on page 105) and the original default duration of the animation (in the 'duration' field). The X3D *Interpolator* nodes are only used as data containers for key-value pairs wrapped by *AnimationContainer* nodes. Here, the order in which the multi-fields 'interpolators', 'fieldnames', and 'targetnames' appear is important: the first interpolator corresponds to the first entries in the fields 'targetnames' and 'fieldnames' and so on.

```
TimedAnimationContainer : X3DAnimationContainer {
  SFString [] name          ""
  MFString [] targetnames   []
  MFString [] fieldnames    []
  MFNode   [] interpolators []
  SFFloat  [] duration      0
}
```

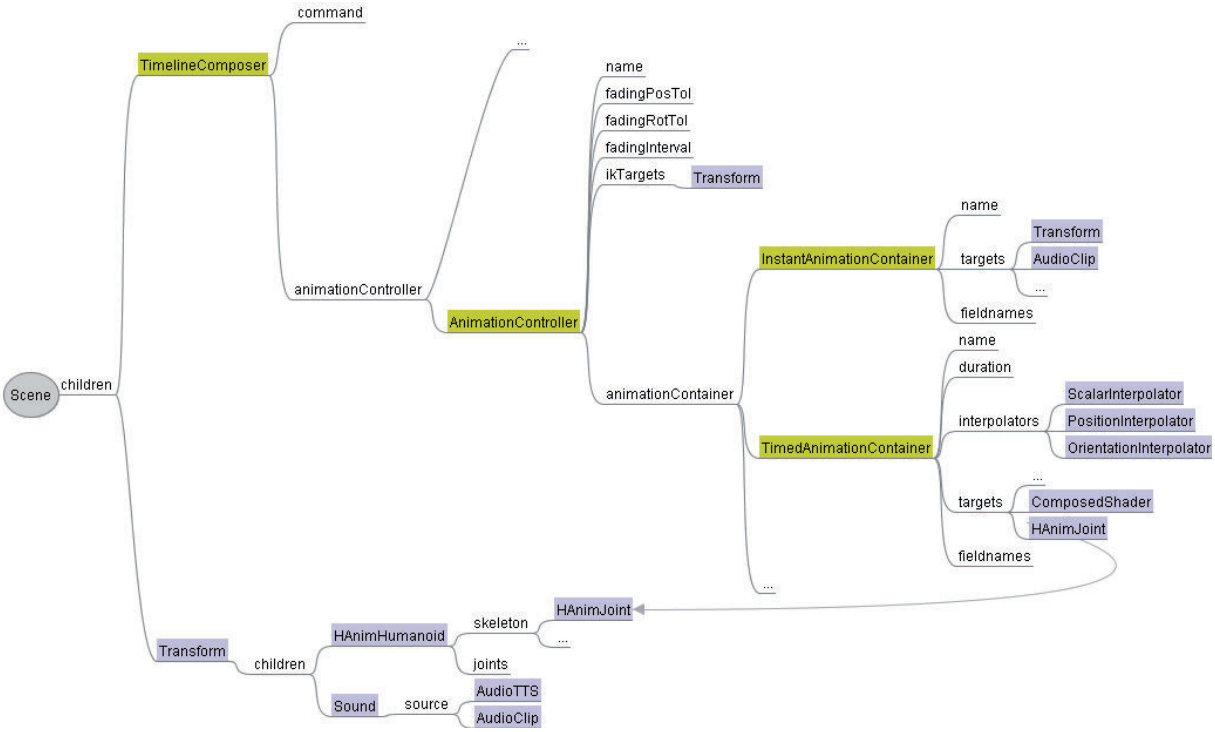

Figure 7.10: *Overview of the proposed system architecture and scene-graph integration. Whenever the TimelineComposer receives a PML command, all requests are processed and forwarded to the responsible AnimationController nodes.*

Whereas the *TimedAnimationContainer* denotes actions with a certain duration, the *InstantAnimationContainer* denotes simple state changes that have no duration (i.e. events or transitions with dur='0'). Examples are events like show, hide, start or stop that are referenced by a `<transform>` in an actions script or a `<fragment>` in the definitions script respectively. The node therefore does not contain interpolators but instead it can hold the id of a media-object as defined in a definitions script such as `<audio>` and `<video>` for referencing additional multimedia data to be integrated into the 3D scene, or GUI elements like `<menu>` and `<slider>` for incorporating user interaction, too.

```
InstantAnimationContainer : X3DAnimationContainer {
  SFString []         name           ""
  MFString []         targetnames    []
  MFString []         fieldnames     []
  SFString [in, out] mediaId        ""
  SFInt32  [out]      changeUIValue
  SFInt32  [in]       receiveUIValue
}
```

Depending on the visualization type attribute ("2D" or "3D") of the requested user interface element, the GUI can be internally created on definition during runtime in case of "2D" (whereas for "3D" specific asset is required). Then, the referenced `<guiContainer>` is assigned an interactive texture node, or more specifically an *UITexture* node as de-

scribed in more detail in [161], which is automatically connected with the 'change-' and 'receiveUIValue' fields of the *InstantAnimationContainer*.

According to the respective UI type, e.g. a slider or radio button menu can be instantiated here. Since our proposed interactive textures utilize standard 2D widget toolkits such as Qt,[3] which can be applied via simple texture mapping to all types of geometry whilst remaining fully functional by mapping mouse and key events onto the X3D pointing sensor concept [336], this way it is very easy to also integrate well-known desktop interaction metaphors into both, X3D and PML, for unified interaction and behavior control.

7.3.5 Examples and Discussion

Obviously each animation container has to handle lots of data, especially if the key times and values were taken from motion capture data. Therefore animation containers can be reloaded during run-time by sending an appropriate definitions script. This is quite convenient, because due to the large amount of data, file sizes soon get too big for handling them in any editor. But this is not only an issue in combination with animation definitions. Currently in X3D the only possibility to gain access to data in other files is via PROTOs and the import/ export mechanism, but the latter is only allowed for ROUTEs between X3D files that are directly inlined, and therefore not viable for our problem.

Consequently another option is to allow node re-USE over file boundaries. In this case the node names may not be unique any more, why we have implemented a slightly different addressing scheme that extends the X3D "Inline" node with an additional field 'nameSpaceName', which allows referencing over file boundaries with the syntax 'nameSpaceName::nodeName'. The question is, in how far this breaks with the concepts of X3D, which have been designed around 15 years ago with web-based applications in mind.

At this time, scene graphs were the only reasonable choice for 3D applications and a typical PC was almost swamped by rendering some geometric primitives. Nowadays, a 3D scene has much more content and lots of interacting elements that cannot be kept in a single file. By allowing references into other files, scene description and handling gets much clearer, but for usage in web environments one has to make sure, that only such nodes can be used, which explicitly may be shared by other people [149].

7.3.5.1 Using the Scripting Interface...

Due to the mentioned drawbacks concerning timing, synchronization, and the lack of an easy to use TTS system, in X3D based applications the output of spoken text is often still done with the help of on-screen displays instead of using audio output. In the following we discuss an example that demonstrates how this task can be simplified with our approach [147, 149]. We begin by describing the high level scripting interface, before explaining the corresponding X3D node extensions. First, the phoneme to viseme mapping and all required animations should be defined. In this regard, the `<createSinglePose>` element helps to create a new `<singlePose>` element from existing elements to achieve more sophisticated visemes or base emotions (see Figure 2.6).

[3] Qt – cross-platform application and UI framework. http://qt.nokia.com/

```
<definitions id="def1">
  <character id="Valerie" src="Valerie.wrl" root="Trafo">
    <voice id="vVal" refId="Klara16"/>
    <viseme>
      <phoneme id="a" refId="Phon_A" intensity="1.0" />
      ...
    </viseme>
    <singlePose id="Emot_Angry" src="angry.wrl" />
    <singlePose id="Emot_Sad" src="sad.wrl" />
    <singlePose id="Phon_A" src="phon_A.wrl" />
    ...
    <multiPoses id="attract" src="anim1.wrl" dur="4167" />
    <multiPoses id="present" src="anim2.wrl" dur="2733" />
  </character>
</definitions>
```

After that, the animations can be started by routing a filename or string with the PML actions script to the 'command' field of the *TimelineComposer* node (cf. Figures 7.2 and 7.5). Here the values of the 'id' attributes must match with the 'name' fields of the animation nodes (see next section). In this example the virtual character Valerie says something, and concurrently makes the gesture 'attract' (see the `<par>` block below), before doing the gesture 'present' (referenced in the `<seq>` block of the actions script).

```
<actions id='act1' start='true'>
  <character refId='Valerie'>
    <speak id='a'>
      <text>Hello!</text>
    </speak>
    <animate id='b'>
      <multiPoses refId='attract'/>
    </animate>
    <animate id='c'>
      <multiPoses refId='present'/>
    </animate>
  </character>
  <schedule>
    <seq>
      <par>
        <action refId='a' begin='1000' dur='0'/>
        <action refId='b' begin='0' dur='4167'/>
      </par>
      <action refId='c' begin='0' dur='2733'/>
    </seq>
  </schedule>
</actions>
```

As can be seen in the definitions script above, the type of voice for parameterizing the *Voice* node is also defined, which is only useful in combination with the *AudioTTS* node.

When having a closer look at the <speak> tag in the actions script shown next, one can notice, that the duration of this action is zero, because the needed phonemes and their lengths have to be internally calculated by the TTS system, and are not known until then. If lip synchronization shall be achieved by using a standard *AudioClip* node instead, via the tag <audio src='hello.wav'> an audio file must be provided. Additionally a list of phonemes with their respective durations, which have to be computed in advance, has to be declared as sub-tag of the audio tag like this: <phoneme refId='h' dur='100'/>.

Although the first alternative is much easier to use, it has the disadvantage, that the exact duration is not known beforehand, because it is internally calculated during runtime and thus can't be synchronized exactly with other actions. This could be alleviated by providing a higher level of abstraction, where the temporal order is given by using generic alignment attributes instead of a <schedule> block, which is often the case for abstract behavior specification languages like BML [189]. But this not only requires a lot of care in order to avoid invalid states, but it is also not always unambiguously resolvable. The grammar of this element, which again is a CHARACTERACTION, is defined as follows:

```
SPEAK ::= <speak id="ID" ALIGNMENT>
            <text>SPEAKTEXT</text>
            (SPEAKAUDIO)^{0,1}
          </speak>
```

Because PML follows an approach of iterative refinement, some tags and attributes are only useful at certain points in the module pipeline. Whereas higher level modules do their scheduling based on the alignment information, this property is ignored on the player level, where only the concrete timing information given by the schedule block is used. In fact, every action therefore is accompanied by an ALIGNMENT that indicates the temporal order of this action related to other actions. If no other action is relevant, this will be indicated by setting both attributes *alignTo* and *alignType* to "null".

7.3.5.2 ...and the Controlling Component

After having explained the high level interface, we will show how this corresponds to our proposed nodes for animation control, and how they can be used in a concrete setting. The following code fragment in VRML encoding shows exemplarily how to define interpolator based animations in *TimedAnimationContainer* nodes, and how an *AnimationController* node for a character or object can look like. Generally, an *AnimationController* is created internally when receiving a new character or object definition and does not need to be defined explicitly, except one wants to define special settings.

```
AnimationController {
  name "Valerie"
  animationContainer [
    TimedAnimationContainer {
      name "attract"
      interpolators [
        OrientationInterpolator {...},
        ...
```

```
      ]
      duration 4.167
      targetnames [ "Bip01_Spine", ... ]
      fieldnames  [ "rotation", ... ]
    },
    ...
  ]
}
```

As mentioned, the X3D *Interpolator* nodes are only used as data containers for key-value pairs, as depicted in Figures 4.5 (page 105) and 7.8. Thus, there is no need for routes or other difficult to maintain helper structures, because all interpolators, which are active at a given time frame t_i (see Figure 4.4 on page 104), are solely used for the internal calculation of joint transformations etc., in order to have access to all required animation data for combining animations efficiently. This way, both gestures from the example PML script above are automatically cross-faded, while also considering the active idle poses from the first example given in section 7.2 to avoid transition artifacts.

The same goes for blending, if instead animations b and c should be played in parallel here (cf. Figure 4.6, page 106). In addition, this concept is extensible not only concerning the scripting interface, but also the controlling component, because it allows to transparently include more sophisticated schemes for blending animations like the usage of transition motions for motion graphs, as well as those for motion generation, like the parameterizable motions used for locomotion generation explained in section 4.3.2.

7.3.6 Considering Autonomous Behavior

Referring to the different layers of abstraction as depicted in Figure 7.1 we have exemplarily developed an additional X3D Steering component based on the OpenSteer C++ library [259]. Its implementation closely follows Reynolds original paper [261] that focuses on path determination for autonomous agents. The component includes a set of nodes for simulating autonomous agents within a scene [135]. They have the ability to navigate around in their world in a life-like and improvisational manner. By combining predefined behaviors like wander, seek or flee behavior, a variety of crowd-like autonomous systems can be simulated to enliven a scene by animating these objects. Integration into PML is achieved by introducing an additional `<crowd>` element, which references a number of characters or objects, and can also have certain behavior types as explained next.

The *SteeringSystem* node, which is also a special type of *SimulationSystem* node (compare section 4.5.5), contains the so-called vehicles of the steering system. Children of the MFNode "vehicles" field are used for neighbor collision avoidance etc. Additionally, for each type of behavior there is a special *SteeringBehaviour* node for simulating a certain behavior, which returns a steering force for e.g. obstacle collision avoidance, wandering or seek behavior. Thereto, the *SteeringVehicle* also has an MFNode field "behaviours" for holding various behaviors such as the *WanderBehaviour*. Furthermore, the *SteeringVehicle* also provides fields for setting attributes like "mass", "radius" and "speed". As visualized in

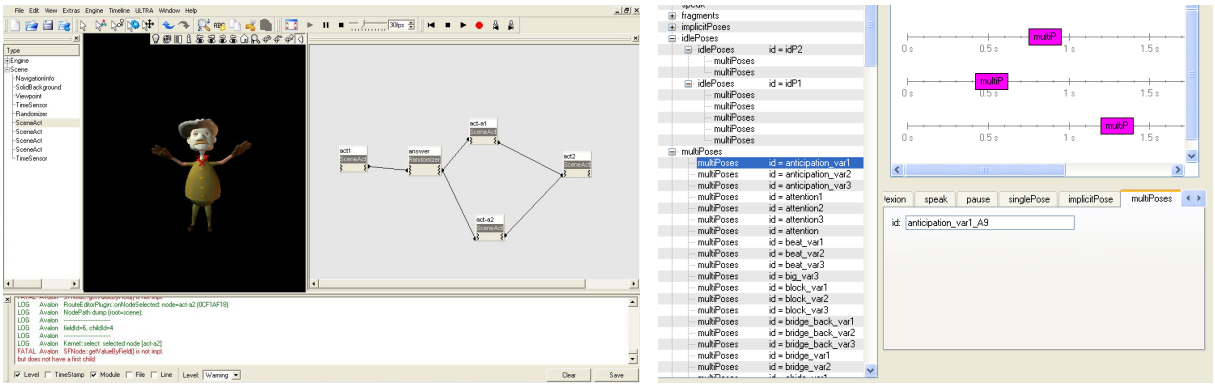

Figure 7.11: *Left: Authoring of story graph with X3D Route editor. Right: PML editor.*

Figure 4.10, the therewith calculated path can then be used to synthesize the appropriate walking motion if necessary.

Likewise more on a proof-of-concept level we have also implemented the *Brain* node for representing the cognitive layer in order to create avatars that can communicate with a user of the X3D world. The node is parameterized with an Eliza-style AIML file [342, 263] for defining the topic. By setting the SFString 'ask' field, one can receive the clear-text answers to the questions sent via the 'answer' out-slot. Albeit the underlying library does not provide means to select appropriate communicative gestures, some typical gesticulations can concurrently provided by an idle list to achieve some more natural talking behavior. But it should be stated, that to our opinion with X3D only the "body", namely the visual and auditive representation, should be modeled, but not the "mind".

7.4 Content Creation and Authoring

Content creation embraces the creation of the character's geometry including materials as well as animations, the scene, the behavior and the overall "story". To create a scene, the content author needs a pool of skills of the used character. As outlined in section 2.1, the creation of the geometry for a virtual character or stage and the accompanying animations (postures, gestures, mimics) is very time consuming and tedious, and it needs talented designers to build 3D models from scratch. Yet, since there is a whole software ecosystem dealing with modeling and animation in a quite sophisticated way, as mentioned that kind of data is assumed to be given and we don't focus on content generation in general but on simplifying the authoring process concerning the temporal alignment of actions.

For the creation of the digital story several methods can be used. Advanced ones provide non-linear storytelling and adaptive dialogs controlled by a dialog manager (compare section 7.3.3), though authoring such dialog engines is not trivial and still takes a lot of effort to get a decent result. But there are many applications where this complexity is not necessary. For such cases we developed simpler tools allowing us to put together story-lines in an easy and intuitive way. As described earlier, a story can be described with PML. It allows defining when and what a character or object in the scene is doing and how or when the user can interact with the virtual environment. The naïve approach

would be to write one big PML script, where the whole story is described. This would allow no interaction by the user and there would be no chance to change the flow of the story in any way. The result would be similar to a film.

Therefore we choose a different approach [183, 156] that can be modeled as a finite state machine $A = (\Sigma, Q, \delta, q_0, F)$ with input alphabet Σ, initial state $q_0 \in Q$ and $F \subseteq Q$ the set of final states. To allow interactions by the user and to change the flow of the story via the transition function δ, we define short acts and transform them into PML scripts.

A short act could be a dialog between two characters on a given topic and is considered a state $q \in Q$ of the story, where the set of all states is denoted by Q. Such PML scripts will be stored in special *SceneAct* nodes, which are connected by routes and can be easily realized as X3D PROTO [336]. The routes define the flow of the story and can be seen as elements of the transition function. As soon as such a node gets a trigger on its input field, a transition occurs and its PML script will be executed. The 'url' field holds the corresponding PML script, whereas the 'run' and 'finished' SFBool slots are used for the story graph composition. The proposed node interface is shown next.

```
SceneAct : X3DAnimationBase {
  SFString [] name ""
  MFString [in,out] url  []
  SFBool   [in]     run
  SFBool   [out]    finished
}
```

By adding other nodes into the route graph, we easily can add some non-linearity and possibilities for user interaction to the story. For example if we want to add randomized answers of a character triggered by a user interaction, we simply could add a *Randomize* node [135] to the route graph. In this case, the state machine is easier modeled as being non-deterministic, i.e. $\delta : Q \times \Sigma \rightarrow Pow(Q)$ maps to a set of subsequent states. The user interaction $a \in \Sigma$ then triggers the *Randomize* node, to whose output field several *SceneAct* nodes are connected via routes. The randomizer chooses one of them and forwards the signal to it, and the associated PML script gets executed.

Beside the technological foundation for the story playback, we also implemented a graphical user interface for putting route/ story graph and the PML scripts together [183, 156]. These GUIs are embedded as plugins into our "Composer" authoring tool for the Instant Reality framework (Figure 7.11, left, visualizes a little story graph, and the right image shows a visual PML editor whose design follows the editor for Adobe Flash [1]).

7.5 Conclusions

In this chapter we have presented a framework for the visualization component of multimodal dialog systems that builds upon the open ISO standard X3D for simplifying the integration of virtual characters into dialog-based systems. This is achieved by a layered approach that introduces an abstraction level on top of X3D by means of a higher level language (i.e. PML), which can be used for module communication and for coordinating

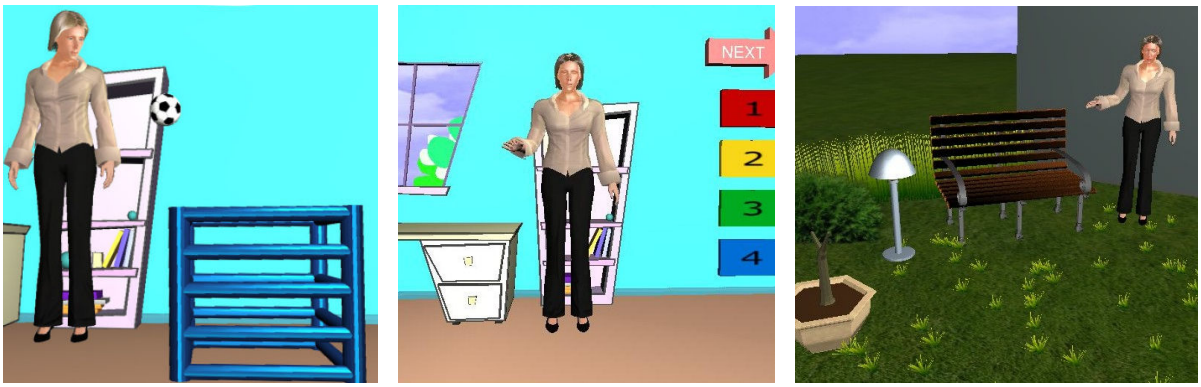

Figure 7.12: *Language learning with virtual characters by responding through solving the tasks.*

the conversational behavior of virtual humans. The proposed framework combines important building blocks from the X3D-based execution layer, which are scalable and generic enough to be also used in different contexts, with a declarative control layer, thereby making all relevant functionalities available in a manageable way, which allows for plausibly reacting virtual characters in dynamic environments.

Furthermore, it was discussed how the control layer is connected with the execution layer, and how it is integrated into X3D and the Instant Reality framework [135] for availability and efficiency and to support a broader range of applications on the one hand as well as into typical software architectures for dialog systems on the other hand. The here discussed control layer is mainly responsible for coordinating and synchronizing animations and events in time. Because this requires flexible control of the character and thus a flexible animation system, we have also outlined how humanoid animation can be efficiently controlled in the context of H-Anim/ X3D [335, 336].

As mentioned in chapter 4, the current H-Anim standard only defines the skeleton setup, whereas definition and handling of animations have never been part of this standard, and the built-in X3D animation mechanisms are not suitable for dealing with multiple animations. In addition, despite its declarative document-based design X3D does not allow for declarative animation and event control. Thus, to overcome some of the limitations concerning animation and authoring, we have proposed a framework for modeling behavior and appearance of dynamic scene elements that is declarative and efficient. The introduced enhancements to the present X3D ISO standard comprise extensions for declaratively controlling animations, which also convert a time schedule, mix animations, and consider resultant dependencies that may need to be simulated.

Moreover, we have presented the scripting and interface language PML that hides complexity and makes the scripting of behavior easier, and thus allows the implementation of story-lines at a higher level, allowing application developers to create and author realistic and interactive 3D environments easily. For one thing, PML differs from other multimodal markup languages in the way messages are used to synchronize modules and to inform them about system and user actions. And for another, it differs in its separation of actions and definitions. The latter not only encapsulate the knowledge about scene elements, but they are also used to locate the resources associated with a character or object action, which helps separating between abstract behavior classes and concrete assets.

Figure 7.13: *Characters scripted via PML for storyboarding and early pre-visualization [160].*

In this regard we also described how the rendering framework can be integrated into module chains like the Virtual Human platform [321], which thereby also supports highly complex scenarios. Hence, besides visual plausibility and data quality also integration aspects must be considered. A clean API design and understanding of each module is crucial, and PML allows defining the interface between the computer graphics and the AI modules, without the need for the AI people to take care of graphics issues and vice versa. However, since current systems still all differ in their API and functionality, PML is abstract enough to be used as interface in different environments and contexts.

Furthermore, we presented our approach for an easy deployment of virtual characters in scenarios of medium complexity by describing a method to build simple interactive, non-linear stories. Here, another example besides dialog-like applications could be the creation of content like the instructional films shown in the airport at the check-in counter or later in the plane. If, comparable to the EMBRScript animation layer described in [123, 180], also procedural aspects and the program logic shall be handled by the presentation component, this can be achieved by combining PML for animation scripting with standard X3D scripting methods, using either the *Script* node or the SAI interface.

The proposed extensions were used and evaluated in different scenarios, like in the so-called ZAMB application developed cooperatively with researchers in the field of AI within the Virtual Human project (as shown in Figures 1.3 and 7.6) that aimed at the development of virtual characters as personal dialog partners. The developed infotainment application, where the users act as coach of the German football team, provides several multimodal interfaces such as speech recognition and 3D GUIs for interacting with the characters. The game itself consists of two stages, where the users discuss the team selection with the virtual experts [183, 156].

Another demonstration from the industrial area utilized a virtual character as a multimodal automotive assistant that acts as the dialog interface for explaining usage and features of a new car, giving status messages, or helping with problems. However, the special requirements of embedded systems are another issue here, since in-car systems are rather limited in their memory and processing power, and even modern graphics hardware for such devices is only Shader Model 2.0 compliant, if at all.

The e-learning application depicted in Figure 7.12 shows a language learning scenario that aims at learning English as a foreign language and is designed for ten years old. Here, learning occurs in direct interaction with a responsive virtual character, which tells the student what to do and also serves as a guide. The learning experience is embedded into story to provide context, structure, and background information. Interactive elements are integrated for "grasping" the learning matter. For instance in Figure 7.12 (left) a scene is shown, where the character asks the child to put the ball into the basket, what can only be fulfilled correctly when understanding the language [146].

Figure 7.13 shows screenshots taken from a scenario currently being developed within the ANSWER project [6]. Albeit here context and objectives are a bit different (namely to provide interactive tools for storyboarding and early pre-visualization for directors and other creative people like game designers), almost the same techniques apply due to the project's top-down approach [160]. And even more, besides the need for having flexible animation and scene control via a declarative XML interface, here camera and lighting aspects are essential for the final perception of a scenario.

Since especially for the latter application type the creative freedom is important, the animation system for instance should be able to synthesize different specificities of a motion, e.g. based on the so-called Effort Shape theory for visualizing various personalities and moods as described in [41]. But the same goes for more advanced interactive dialogs, too. Also, for future developments the path planning component should be extended, which is currently only rudimentary implemented, whilst also focusing on motion-graph-based motion synthesis, and thereby the development of more advanced animation generators for also handling goal-directed motions in general.

Likewise the proposed animation scripting language PML, which currently focuses on the specification of verbal and non-verbal behaviors of virtual characters in multi-party dialogs, could be generalized for a broader range of applications, e.g. with a more sophisticated way of controlling lighting etc. In addition, the authoring system mentioned in the last section could be further extended and an object and animation library should be build up. In this regard, it would also be helpful to connect the avatar and object geometries with semantics to allow for intelligent objects that for instance can be used as trigger for a script, e.g. by utilizing the smart objects paradigm described in [109].

8 Conclusion and Future Work

The ultimate objective of this work is to have virtual characters as personal and believable dialog partners in multimodal dialogs. This does not only require a reliable and consistent motion and dialog behavior that also regards nonverbal communication and affective components, but also efficiency for the realization of applications. Besides the creation of context-dependent, intelligent communication behavior, which is considered an AI problem, the corresponding presentation or "surface realizations" [123], such as gestures and mimics, belong to the domain of CG and hence were subject of this research.

8.1 Summary and Conclusion

More specifically, in this thesis dynamic aspects of character rendering in the context of multimodal dialog systems were described. Therefore, a system for the visualization component of such multimodal interfaces was defined and developed, that allows embedding virtual characters into complex 3D environments, whereas the typical scope of research on the presentation layer, namely character animations and audio playback [180], was only handled where necessary. Though the focus clearly lies on the graphical representation, a high degree of control and flexibility must be provided, too.

In this regard, the framework presented in this thesis builds upon established visualization techniques and open standards such as X3D [336], a 3D file format that describes the functional behavior of time-based and interactive 3D contents and is also easy to learn for non-programmers. The proposed system integrates all relevant low-level functionalities within a modular, scene-graph-based architecture. Most building blocks are furthermore made accessible to behavior-generating systems via a declarative interface and control layer to allow for fully responsive characters in interactive virtual environments. This relationship conceptually was illustrated in Figure 1.4 on page 28.

Therefore, simplifying the deployment and integration of virtual characters into dialog-based systems is firstly achieved by a layered approach that introduces another level of abstraction on top of the scene-graph by means of the higher level interface and control language PML. As was discussed in chapter 7, this language can be used for directly scripting animations or for module communication and coordinating the conversational behavior of virtual humans. To bridge the gap between the presented layers also scene-graph nodes for controlling animations were introduced that are able to convert the scripted schedules and to mix an arbitrary number of animations, whilst still being extensible to new concepts of motion generation (compare section 7.3.4).

Further, connected subquestions were worked out such as character animation and hair simulation (see chapter 4). Simulating plausible behavior not only requires flexible con-

trol of the character and thus a flexible animation system, but also the consideration of resultant dependencies that need to be simulated. Consequently, adjoint effects like moving hair, which must be simulated if body animations were generated or modified during runtime, were also integrated while considering the duality of simulation and rendering especially for GPU-based methods. In this regard, a self-contained runtime system was described with matching techniques and building blocks (e.g. nodes for simulating hair or droplet flow) on the scene-graph-based execution layer.

Although a lot of work already is done towards more realism, the target of research usually is conducted in a single standalone tool without embedding the algorithms proposed into a wider field of applications. Additionally, modern rendering techniques well-known from movies and computer games are mostly ignored in the fields of VR/AR and multimodal dialog systems, since here communicative effectiveness is often optimized by using unrealistic behaviors. Also, appearance or camera work are mostly ignored, albeit they help to correctly perceive the communicative intent. To tackle these issues, extrinsic factors were incorporated, like the rendering system itself, properties of human skin and hair, as well as lighting and shading in general, for also supporting Mixed Reality applications that "fuse" real and virtual worlds.

Furthermore, in chapter 6 several enhancements to the current X3D standard were presented, which allow utilizing it not only for standard desktop VR but also for Augmented and Mixed Reality applications alike. This is especially of importance when trying to integrate virtual characters as man-machine interface into Mixed Reality applications with their special requirements in terms of hardware and realism. The latter requires a consistent rendering between real and virtual parts, which implies having suitable methods for lighting reconstruction and simulation for a seamless integration (cf. section 6.4). However, depending on the type of MR application using embodied agents can be more efficient than standard WIMP-like interaction.

Moreover, a rather unattended field beyond standard joint- and mesh-based animations are psycho-physiological processes like crying, blushing, or turning pale. These likewise are part of the nonverbal communication but until recently were mostly left unconsidered. However, these symptoms are not only important, because in dialog systems often a close-up view of the character's face is shown, but they are also essential for the correct perception of some emotions in the context of nonverbal communication. Hence, besides supporting standard animation methods as vital requirement for a dialog system, other intrinsic factors for modeling behavioral aspects were considered, too, including rendering aspects of emotions and their appropriate control mechanisms.

Therefore, techniques for creating skin tone changes based on a novel parameterizable emotion model were presented in chapter 5. To define this model, a classification of emotionally caused skin changes based on physiological and psychological knowledge was performed and evaluated. Furthermore, an on-surface droplet flow simulation was integrated for simulating weeping (see section 5.3). Since these methods can be executed in real-time, they can be combined with facial animations in a consistent manner to present visually convincing emotions for virtual characters.

In addition, a declarative approach to camera placement was presented in section 6.2, which utilizes well-known guidelines from the film area and allows framing virtual charac-

ters and others objects in an intuitive manner by defining where on the screen an object shall appear. The cinematographic camera approach thereby not only helps with coordinating graphics and language, but also with framing subtle effects such as blushing and crying that might otherwise be lost in the big picture. This includes rendering techniques to present these effects in high quality and to enhance the impression.

Thus, the discussed approach offers enduring solutions and more efficiency by means of the integration into well established visualization techniques like the scene-graph (where the proposed building blocks are scalable and generic enough to be useful for other purposes), into existing open standards like X3D and H-Anim [336, 335] as well as into the Mixed Reality system Instant Reality (which utilizes X3D as application description language [135]) for targeting a broader range of applications, and lastly into more abstract system architectures like the typical module pipelines as used in the ECA community.

Finally, the applicability of the proposed concepts and techniques was proven and evaluated in different fields of applications, demonstrating interactive manuals, infotainment and edutainment applications, cultural heritage scenarios as well as early pre-visualization and the like. In this regard, expressive virtual characters also allow for novel evaluation possibilities, e.g. in combination with the SHORE library [197], which not only provides face (and eye) detection and tracking, but also a classification of age, gender, and facial expressions, and which is integrated into Instant Reality's Computer Vision subsystem. Moreover, first tests with Microsoft's new *Kinect* sensor seem very promising. For one thing, the depth image comes for free, which directly allows occlusion handling in AR scenes, and for another, the device can be used as a cheap means for motion capturing.

8.2 Future Work

Albeit various issues could be overcome with the concepts presented in this thesis, there are still many open problems that are left for future work, since not all aspects could be considered here. Hence, a variety of possibilities exist to improve the current state of the proposed system. For example, the industrial application shortly mentioned in the last chapter has shown that having optimized workflows including asset assembly is a key issue for the wide adoption of virtual characters. Therefore, an authoring system that allows developing X3D-/ PML-based applications in a graphically interactive manner, whilst also providing a comprehensive object and animation library, would be advantageous especially in those cases, where the system is used without the high-level AI components.

In addition, both integrated methods for generating motions during runtime, either procedurally or using parameterizable motions, though no core topics of this thesis, are only proof-of-concept implementations and need further developments especially regarding more expressivity. However, due to several competing requirements, mainly realism, real-time capability, and parameterability, this is still a field of active research and many existing approaches are not yet real-time capable. Also, a more sophisticated coarticulation scheme for lip synchronization would be desirable here.

Furthermore, the presented hair simulation should be able to deal with other hair styles such as curls. In this regard one should keep in mind, that the development cycles of

GPUs are very short at the moment – lately OpenGL 4.1 with Shader Model 5.0 was released. This opens up new possibilities for rendering and simulating hair (compare e.g. [352]) and similar deformable objects. Another open issue is the fact that the SH-based lighting model is not able to cope with high frequencies and that the currently utilized transfer functions are too simple. Moreover, in order to avoid relighting errors, a faster and more robust method for material reconstruction is necessary.

Incorporating the introduced building blocks into the X3D standard would enable a further proliferation and acceptance. Hence, there are currently ongoing efforts within the Web3D Consortium to reconcile some of the aforementioned approaches towards their standardization. One could also think of including more abstract cinematographic concepts, such as an over-the-shoulder shot. Because the latter already requires semantic knowledge, it would be helpful to connect the geometric data with semantics. Finally, an in-depth evaluation of the proposed model for classifying and parameterizing emotionally caused skin changes in collaboration with psychologists and physicians is eligible.

And as was shown with James Cameron's latest movie "Avatar", in this respect the so-called uncanny valley now can be overcome with enough money, time, and manpower. However, even though the virtual characters seem life-like, they are aliens with blue skin, stylized faces, different limbs, and they live in a low-gravity world. But besides that obviously increasing interest in character technology, this film production also reveals the main challenges in this area in the coming years, since the demonstrated technologies here are far from real-time, and the dialogs are not interactive at all.

Bibliography

[1] Adobe. Flash, 2010. http://www.adobe.com/products/flashplayer/.

[2] Akenine-Möller, T., Haines, E., and Hoffmann, N. *Real-Time Rendering*, 3 ed. AK Peters, Wellesley, MA, 2008.

[3] Alexa, M. Linear combination of transformations. *ACM Trans. Graph. 21*, 3 (2002), 380–387.

[4] Alexa, M., Behr, J., and Müller, W. The morph node. *Web3D - VRML 2000 Proceedings* (2000), 29–34.

[5] Amor, H. B., Heumer, G., Jung, B., and Vitzthum, A. Grasp synthesis from low-dimensional probabilistic grasp models. *Journal of Visualization and Computer Animation 19*, 3-4 (2008), 445–454.

[6] ANSWER. Artistic-Notation-based Software Engineering for Film, Animation and Computer Games, 2010. http://www.answer-project.org.

[7] Arafa, Y., Kamyab, K., and Mamdani, E. Character animation scripting languages: a comparison. In *AAMAS '03: Proc. of the 2nd int. joint conference on Autonomous agents and multiagent systems* (NY, USA, 2003), ACM, pp. 920–921.

[8] ARB. GL_ARB_occlusion_query, 2003. OpenGL Extension Registry.

[9] ARB. GL_ARB_framebuffer_object, 2008. OpenGL Extension Registry.

[10] Arnaud, R., and Barnes, M. *Collada*. AK Peters, 2006.

[11] Aschenberner, B., and Weiss, C. Phoneme-viseme mapping for german: Video-realistic audio-visual-speech-synthesis. Tech. Rep. IKP - Working Paper NF 11, Universität Bonn, Institut für Kommunikationsforschung und Phonetik, 2005.

[12] Autodesk. 3ds max 2011, 2010. http://area.autodesk.com/3dsmax2011/features.

[13] Autodesk. Autodesk FBX: Platform-independent 3d data interchange technology, 2010. http://www.autodesk.com/fbx/.

[14] Autodesk. Maya 2011, 2010. http://area.autodesk.com/maya2011/features.

[15] Babski, C., and Thalmann, D. 3d on the web and virtual humans, 2000.

[16] Badler, N. I., Phillips, C. B., and Webber, B. L. *Simulating Humans: Computer Graphics, Animation, and Control*. Oxford University Press, 1994.

[17] Bando, Y., Chen, B.-Y., and Nishita, T. Animating hair with loosely connected particles. *Computer Graphics Forum 22*, 3 (2003), 411 – 418.

[18] Baraff, D., and Witkin, A. Large steps in cloth simulation. *Computer Graphics 32* (1998), 43–54.

[19] Barakonyi, I., and Schmalstieg, D. Ubiquitous animated agents for augmented reality. In *ISMAR '06: Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 145–154.

[20] Beales, R. M., Chakravarthy, A., Hedtke, R., Huther, W., Jung, C., Jung, Y., Koutsoutos, S., and Yannopoulos, A. Automated 3d pre-vis for modern production. In *International Broadcasting Convention (IBC) 2009. Conference Publication: Technical Papers.* (London, 2009). 8 p.

[21] Becker-Asano, C., and Wachsmuth, I. Affective computing with primary and secondary emotions in a virtual human. *Autonomous Agents and Multi-Agent Systems 20*, 1 (2010), 32 – 49.

[22] Behr, J. *Avalon: Ein skalierbares Rahmensystem für dynamische Mixed-Reality Anwendungen.* Dissertation, TU Darmstadt, 2005.

[23] Behr, J., Dähne, P., Jung, Y., and Webel, S. Beyond the web browser - X3D and immersive VR. In *IEEE VR 2007: VR Tutorial and Workshop Proceedings: IEEE Symposium on 3D UI* (Piscataway, USA, 2007), IEEE.

[24] Behr, J., Dähne, P., and Roth, M. Utilizing X3D for immersive environments. In *Web3D '04: Proc. of the ninth int. conf. on 3D Web technology* (NY, USA, 2004), ACM Press, pp. 71–78.

[25] Bertails, F., Audoly, B., Cani, M.-P., Querleux, B., Leroy, F., and Lévêque, J.-L. Super-helices for predicting the dynamics of natural hair. *ACM Trans. Graph. 25*, 3 (2006), 1180–1187. Special issue: SIGGRAPH '06.

[26] Bitmanagement. Drawgroup & drawop, 2002. http://www.bitmanagement.de/developer/contact/examples/multitexture/drawgroup.html.

[27] Bleser, G., Wuest, H., and Stricker, D. Online camera pose estimation in partially known and dynamic scenes. In *ISMAR 2006 : Proc. of the Fourth IEEE and ACM Int. Symposium on Mixed and Augmented Reality* (Los Alamitos, Calif., 2006), IEEE Computer Society, pp. 56–65.

[28] Blinn, J. Where am I? What am I looking at? *IEEE Computer Graphics and Applications 22* (1988), 179–188.

[29] Blinn, J. F. Models of light reflection for computer synthesized images. 192–198. Computer Graphics Vol 11(2).

[30] Blinn, J. F. Simulation of wrinkled surfaces. 1978, pp. 286–292. SIGGRAPH 78.

[31] Borshukov, G., and Lewis, J. P. Realistic human face rendering for "the matrix reloaded". In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches & Applications* (New York, NY, USA, 2003), ACM, p. 1.

[32] Buisine, S., Wang, Y., and Grynszpan, O. Empirical investigation of the temporal relations between speech and facial expressions of emotion. *Journal on Multimodal User Interfaces* (2010), 1–8.

[33] Buttussi, F., Chittaro, L., and Nadalutti, D. H-animator: a visual tool for modeling, reuse and sharing of x3d humanoid animations. In *Web3D '06: Proc. of the 11th int. conf. on 3D web technology* (NY, USA, 2006), ACM, pp. 109–117.

[34] Cal3d, 2006. http://home.gna.org/cal3d/.

[35] Canutescu, A. A., and Dunbrack, R. L. Cyclic coordinate descent: A robotics algorithm for protein loop closure. *Protein Science 12* (2003), 963–972.

[36] Carstensen, K.-U., Ebert, C., Ebert, C., Jekat, S., Klabunde, R., and

Langer, H. *Computerlinguistik und Sprachtechnologie – Eine Einführung*, third ed. Spektrum, Heidelberg, 2010.

[37] Cerekovic, A., Pejsa, T., and Pandzic, I. S. Realactor: Character animation and multimodal behavior realization system. In *IVA '09: Proceedings of the 9th International Conference on Intelligent Virtual Agents* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 486–487.

[38] Chandrasiri, N. P., Naemura, T., Ishizuka, M., Harashima, H., and Barakonyi, I. Internet communication using real-time facial expression analysis and synthesis. *IEEE MultiMedia 11*, 3 (2004), 20–29.

[39] Chang, J. T., Jin, J., and Yu, Y. A practical model for hair mutual interactions. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/ Eurographics symposium on Computer animation* (2002), ACM Press, pp. 73–80.

[40] Charamel. Charactor sdk, 2009. http://www.charamel.com/en/technology/.

[41] Chi, D., Costa, M., Zhao, L., and Badler, N. The EMOTE model for effort and shape. In *SIGGRAPH '00* (NY, USA, 2000), ACM Press, pp. 173–182.

[42] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, 2005.

[43] Christianson, D. B., Anderson, S. E., wei He, L., Salesin, D., Weld, D. S., and Cohen, M. F. Declarative camera control for automatic cinematography. In *AAAI/IAAI, Vol. 1* (1996), pp. 148–155.

[44] Christie, M., and Olivier, P. Camera Control in Computer Graphics. In *Eurographics 2006 - State of the Art Reports* (2006), Eurographics Association, pp. 89–113.

[45] Christie, M., and Olivier, P. Camera control in computer graphics: models, techniques and applications. In *SIGGRAPH ASIA '09: ACM SIGGRAPH ASIA 2009 Courses* (New York, NY, USA, 2009), ACM, pp. 1–197.

[46] Conway, M., Audia, S., Burnette, T., Cosgrove, D., and Christiansen, K. Alice: lessons learned from building a 3d system for novices. In *CHI '00: Proceedings of SIGCHI conference on Human factors in computer systems* (NY, USA, 2000), ACM, pp. 486–493.

[47] Courgeon, M., Martin, J.-C., and Jacquemin, C. Marc: a multimodal affective and reactive character. In *Proceedings of the Workshop on Affective Interaction on Natural Environment (AFFINE)* (2008).

[48] Crow, F. C. Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph. 11*, 2 (1977), 242–248.

[49] Dachsbacher, C., and Stamminger, M. Translucent shadow maps. pp. 197–201. Proceedings of Eurographics Symposium on Rendering.

[50] Dachselt, R., and Rukzio, E. Behavior3d: an xml-based framework for 3d graphics behavior. In *Web3D '03: Proc. of the 8th int. conf. on 3D Web technology* (NY, USA, 2003), ACM, pp. 101–112.

[51] Dähne, P. *Entwurf eines Rahmensystems für mobile Augmented-Reality-Anwendungen*. PhD thesis, TU Darmstadt, Darmstadt, 2008.

[52] Dähne, P., and Seibert, H. Managing Data Flow in Interactive MR Applications. *WSCG* (2005), 173–176.

[53] Daldegan, A., Magnenat-Thalmann, N., Kurihara, T., and Thalmann, D. An integrated system for modeling, animating and rendering hair. *Computer Graphics Forum 12*, 3 (1993), 211 – 221.

[54] De Carolis, B., Pelachaud, C., Poggi, I., and Steedman, M. Apml, a mark-up language for believable behavior generation. In *Embodied conversational agents - let's specify and evaluate them! Proc. of AAMAS Workshop* (2002).

[55] de Carvalho, G. N. M., Gill, T., and Parisi, T. X3d programmable shaders. In *Web3D '04: Proceedings of the ninth international conference on 3D Web technology* (New York, NY, USA, 2004), ACM, pp. 99–108.

[56] de Melo, C., and Paiva, A. Expression of emotions in virtual humans using lights, shadows, composition and filters. In *ACII '07: Proceedings of the 2nd international conference on Affective Computing and Intelligent Interaction* (Berlin, Heidelberg, 2007), Springer-Verlag, pp. 546–557.

[57] de Melo, C. M., and Gratch, J. Expression of emotions using wrinkles, blushing, sweating and tears. In *Intelligent Virtual Agents: 9th International Conference, IVA 2009* (Heidelberg, 2009), Springer, pp. 188–200.

[58] Debevec, P. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proc. of SIGGRAPH 98* (1998), CG Proc., Annual Conf. Series, pp. 189–198.

[59] Debevec, P. Light probe image gallery, 2004. http://www.debevec.org/Probes/.

[60] Debevec, P. A median cut algorithm for light probe sampling. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Posters* (New York, USA, 2005), ACM, p. 66.

[61] Debevec, P. E., and Malik, J. Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH 97* (1997), CG Proc., Annual Conference Series, pp. 369–378.

[62] del Puy Carretero, M., Oyarzun, D., Ortiz, A., Aizpurua, I., and Posada, J. Virtual characters facial and body animation through the edition and interpretation of mark-up languages. *Computers & Graphics 29*, 2 (2005), 189–194.

[63] Delepoulle, S., Renaud, C., and Chelle, M. Improving light position in a growth chamber through the use of a genetic algorithm. In *Artificial Intelligence Techniques for Computer Graphics*. Springer-Verlag, 2009, ch. 5, pp. 67–82.

[64] d'Eon, E., and Luebke, D. Advanced techniques for realistic real-time skin rendering. In *GPU Gems 3*. Addison-Wesley, 2007, ch. 14, pp. 293–347.

[65] Diefenbach, P. J., and Badler, N. I. Multi-pass pipeline rendering: realism for dynamic environments. In *I3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics* (New York, NY, USA, 1997), ACM, pp. 59–70.

[66] Dijk, C., de Jong, P., and Peters, M. The remedial value of blushing in the context of transgressions and mishaps. *Emotion 9*, 2 (2009), 287–291.

[67] Donnelly, W., and Lauritzen, A. Variance shadow maps. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, USA, 2006), ACM, pp. 161–165.

[68] Doswell, J. T. It's virtually pedagogical: pedagogical agents in mixed reality learning environments. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Educators program* (New York, USA, 2005), ACM, p. 25.

[69] Drettakis, G., Robert, L., and Bougnoux, S. Interactive common illumination for computer augmented reality. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97* (London, UK, 1997), Springer-Verlag, pp. 45–56.

[70] Eberly, D. H. *Game Physics*. Morgan Kaufmann Publishers, San Francisco, 2004.

[71] Eberly, D. H. *3D Game Engine Design, Second Edition: A Practical Approach to Real-Time Computer Graphics (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[72] Edsall, J. Animation blending: Achieving inverse kinematics and more. Gamasutra, 2003. http://www.gamasutra.com/features/20030704/edsall_01.shtml.

[73] Egges, A. *Real-time Animation of Interactive Virtual Humans*. Dissertation, University of Geneva, 2006.

[74] Egges, A., Papagiannakis, G., and Magnenat-Thalmann, N. Presence and interaction in mixed reality environments. *Vis. Comput. 23*, 5 (2007), 317–333.

[75] Egges, A., Visser, R., and Magnenat-Thalmann, N. Example-based idle motion synthesis in a real-time application. In *Proc. CapTech Workshop on Modelling and Motion Capture Techniques for Virtual Environments* (2004), pp. 13–19.

[76] Ekman, P. *Gesichtsausdruck und Gefühl: 20 Jahre Forschung*. Junfermannsche Verlagsbuchhandlung, Paderborn, Germany, 1988.

[77] Ekman, P., and Friesen, W. V. The facial action coding system. *Consulting Psychologists' Press* (1978).

[78] Ekman, P., and Rosenberg, E. *What the Face Reveals: Basic and Applied Studies of Spontaneous Expression Using the Facial Action Coding System (FACS)*. Oxford University Press, Oxford, 2005.

[79] Ellis, P. M., and Bryson, J. J. The significance of textures for affective interfaces. 394–404.

[80] Engel, K., Hadwiger, M., Kniss, J. M., Rezk-Salama, C., and Weiskopf, D. *Real-time volume graphics*. A K Peters, Ltd., Wellesley, MA, 2006.

[81] Fernando, R., Ed. *GPU Gems*. Addison Wesley, 2004.

[82] Fernando, R. Percentage-closer soft shadows. In *SIGGRAPH '05: Sketches* (New York, USA, 2005), ACM Press, p. 35.

[83] Fernando, R., and Kilgard, M. J. *The Cg Tutorial*. Addison Wesley, 2003.

[84] Fernando, R., and Kilgariff, E. The GeForce 6 Series GPU Architecture. In *GPU Gems 2*, M. Pharr, Ed. Addison Wesley, 2005, ch. 14, pp. 471–491.

[85] Ferreira, F. P., Gelatti, G., and Musse, S. R. Intelligent virtual environment and camera control in behavioural simulation. In *SIBGRAPI '02: Proc. of the 15th Brazilian Symp. on CG and Image Proc.* (Washington, DC, USA, 2002), IEEE, pp. 365–372.

[86] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. *Computer Graphics Principles and Practice*, second ed. Addison-Wesley, 1997.

[87] François, G., Gautron, P., Breton, G., and Bouatouch, K. Anatomically accurate modeling and rendering of the human eye. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches* (New York, USA, 2007), ACM, p. 59.

[88] Franke, T., and Jung, Y. Precomputed radiance transfer for X3D based mixed reality applications. In *Proceedings Web3D 2008: 13th International Conference on 3D Web Technology* (New York, USA, 2008), S. Spencer, Ed., ACM Press, pp. 7–10.

[89] Franke, T., and Jung, Y. Real-time mixed reality with GPU techniques. In *INSTICC: GRAPP 2008: Proc. of the 3rd Int. Conf. on Computer Graphics Theory and Applications* (Setubal, 2008), INSTICC Press, pp. 249–252.

[90] Fuhrmann, A., Sobottka, G., and Gross, C. Distance fields for rapid collision detection in physically based modeling. In *Graphicon '03* (2003), pp. 58–64.

[91] Gebhard, P. ALMA - a layered model of affect. In *Proc. of the Fourth Int. Joint Conference on Autonomous Agents and Multiagent Systems* (2005), pp. 29–36.

[92] Gerlach, E., and Grosse, P. *Physik: eine Einführung für Ingenieure*, 3 ed. B. G. Teubner, Stuttgart, 1995.

[93] Giacomo, T. D., Moccozet, L., Magnenat-Thalmann, N., Boulic, R., and Thalmann, D. Towards Automatic Character Skeletonization and Interactive Skin Deformation . In *Eurographics 2007 - State of the Art Reports* (Prague, 2007), D. Schmalstieg and J. Bittner, Eds., Eurographics Association, pp. 47–61.

[94] Gibson, S., and Chalmers, A. Photorealistic augmented reality. Eurographics 2003 Tutorial, 2003.

[95] Gibson, S., Cook, J., Howard, T., and Hubbold, R. Rapid shadow generation in real-world lighting environments. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, 2003), Eurographics Association, pp. 219–229.

[96] Gibson, S., Cook, J., Howard, T., and Hubbold, R. Aris: Augmented reality image synthesis. Tech. Rep. IST-2000-28707, University Manchester, 2004.

[97] Gibson, S., Howard, T. J., and Hubbold, R. J. Flexible image-based photometric reconstruction using virtual light sources. *CG Forum (Proc. Eurographics 2001, Manchester, UK) 20(3)* (2001), C203–C214.

[98] Gillies, M., and Spanlang, B. Comparing and evaluating real time character engines for virtual environments. *Presence: Teleoper. Virtual Environ. 19*, 2 (2010), 95–117.

[99] Gleicher, M. Retargeting motion to new characters. In *SIGGRAPH '98* (New York, NY, USA, 1998), ACM, pp. 33–42.

[100] Göbel, S., Schneider, O., Iurgel, I., Feix, A., Knöpfle, C., and Rettig, A. Virtual human: Storytelling and computer graphics for a virtual human platform. In *TIDSE 2004* (Darmstadt, 2004), pp. 79 – 88.

[101] Goldman, D. B. Fake fur rendering. In *SIGGRAPH '97* (1997), ACM Press/ Addison-Wesley Publishing Co., pp. 127–134.

[102] GPGPU. *General-Purpose Computation on Graphics Hardware*, 2010. http://www.gpgpu.org/.

[103] Granny3d, 2010. http://www.radgametools.com/granny.html.

[104] Gratch, J., Rickel, J., Andre, E., Cassell, J., Petajan, E., and Badler, N. Creating interactive virtual humans: some assembly required. *Intelligent Systems, IEEE 17*, 4 (2002), 54–63.

[105] Green, R. Spherical harmonic lighting: The gritty details. *Archives of the Game Developers Conference* (2003).

[106] Green, S. Real-time approximations to subsurface scattering. In *GPU Gems*. Addison Wesley, 2004, ch. 16, pp. 263–278.

[107] Grosch, T. Differential photon mapping: Consistent augmentation of photographs with correction of all light paths. In *Eurographics 2005, EG Short Pres.* (2005), M. Alexa and J. Marks, Eds., pp. 53–56.

[108] Guang, Y., and Zhiyong, H. A method of human short hair modeling and real time animation. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications* (2002), IEEE Computer Society, p. 435.

[109] Gutierrez, M. A., Vexo, F., and Thalmann, D. *Stepping into Virtual Reality*. Springer, London, 2008.

[110] Hachet, M., Dècle, F., Knödel, S., and Guitton, P. Navidget for easy 3d camera positioning from 2d inputs. In *Proceedings of IEEE 3DUI - Symposium on 3D User Interfaces* (2008).

[111] Hadap, S., and Magnenat-Thalmann, N. Modeling dynamic hair as a continuum. *Computer Graphics Forum 20*, 3 (2001).

[112] Hajjar, J.-F. E., Jolivet, V., Ghazanfarpour, D., and Pueyo, X. A model for real-time on-surface flows. *The Visual Computer 25*, 2 (2008), 87–100.

[113] Hamm, A. Psychologie der Emotionen. In *Neuropsychologie*, H. Karnath and P. Thier, Eds. Springer Verlag, Berlin, Heidelberg, 2006, pp. 527–535.

[114] Hammon, E. Practical post-process depth of field. In *GPU Gems 3*, H. Nguyen, Ed. Addison-Wesley, 2007, ch. 28, pp. 583–605.

[115] Hanrahan, P., and Krüger, W. Reflection from layered surfaces due to subsurface scattering. pp. 165–174. Proceedings in SIGGGRAPH 93.

[116] Harris, M. Gpu gems. Addison Wesley, 2004, ch. Fast Fluid Dynamics Simulation on the GPU, pp. 637–665.

[117] Hart, E. Geforce 8800 opengl extensions. NVIDIA, 2007. http://developer.download.nvidia.com/presentations/2007/gdc/G80-OpenGL.pdf.

[118] Hartley, R., and Zisserman, A. *Multiple View Geometry in Computer Vision*. Cambridge Univ. Press, 2000.

[119] Hawkins, B. *Real-Time Cinematography for Games (Game Development Series)*. Charles River Media Inc., Rockland, USA, 2004.

[120] Hayashi, M. TVML (TV program making language). In *SIGGRAPH '98: ACM SIGGRAPH 98 Conference abstracts and applications* (New York, USA, 1998), ACM, p. 292.

[121] He, L.-w., Cohen, M. F., and Salesin, D. H. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *SIGGRAPH '96: Proceedings* (New York, USA, 1996), ACM, pp. 217–224.

[122] Heckbert, P. Introduction to ray tracing. 288–293. Andrew Glassner ed., Academic Press, London.

[123] Heloir, A., and Kipp, M. EMBR – a realtime animation engine for interactive embodied agents. In *Intelligent Virtual Agents: 9th International Conference, IVA 2009* (Heidelberg, 2009), Springer, pp. 393–404.

[124] Hempe, N. Robuste Echtzeitschatten für komplexe, dynamische Szenen. Diplomarbeit, Universität Koblenz-Landau, 2005.

[125] Henyey, L., and Greenstein, J. L. Diffuse radiation in the galaxy. *Astrophysics J. 93* (1941), 70–83.

[126] Hernandez, B., and Rudomin, I. Hair paint. In *CGI '04: Proceedings of the Computer Graphics International* (2004), IEEE Computer Society, pp. 578–581.

[127] Hery, C. Implementing a skin bssrdf, 2003. RenderMan course notes, Siggraph.

[128] Herzog, G., and Reithinger, N. The smartkom architecture: A framework for multimodal dialogue systems. In *SmartKom: Foundations of Multimodal Dialogue Systems*, W. Wahlster, Ed. Springer, Berlin, 2006, pp. 55–70.

[129] Hildenbrand, D. *Geometric Computing in Computer Graphics and Robotics Using Conformal Geometric Algebra*. Dissertation, TU Darmstadt, 2006.

[130] Huang, Z., Eliëns, A., and Visser, C. Implementation of a scripting language for vrml/x3d-based embodied agents. In *Web3D '03: Proc. of the 8th int. conf. on 3D Web technology* (NY, USA, 2003), ACM, pp. 91–100.

[131] ichi Anjyo, K., Usami, Y., and Kurihara, T. A simple method for extracting the natural beauty of hair. In *SIGGRAPH '92* (New York, USA, 1992), ACM, pp. 111–120.

[132] Ieronutti, L., and Chittaro, L. A virtual human architecture that integrates kinematic, physical and behavioral aspects to control h-anim characters. In *Web3D '05: Proc. of the 10th int. conf. on 3D Web technology* (NY, USA, 2005), ACM, pp. 75–83.

[133] Igarashi, T., Nishino, K., and Nayar, S. The appearance of human skin. Tech. rep., Department of Computer Science, Columbia University, 2005.

[134] Innoventive. Frameforge, 2010. http://www.frameforge3d.com/.

[135] IR. Instant Reality, 2010. http://www.instantreality.org/.

[136] Iurgel, I. A. Emotional interaction in a hybrid conversation group. In *International Workshop on Lifelike Animated Agents. Working Notes in Proceedings PRICAI-02* (Tokyo, Japan, 2002), pp. 52–57.

[137] Jacobs, K., and Loscos, C. Classification of Illumination Methods for Mixed Reality. In *Eurographics 2004 - State of the Art Reports* (2004), Eurographics Association, pp. 95–118.

[138] Jensen, H. W., Marschner, S. R., Levoy, M., and Hanrahan, P. A practical model for subsurface light transport. In *SIGGRAPH '01* (New York, USA, 2001), ACM, pp. 511–518.

[139] Jimenez, J., Whelan, D., Sundstedt, V., and Gutierrez, D. Real-time realistic skin translucency. *Computer Graphics and Applications, IEEE 30*, 4 (2010), 32–41.

[140] Jinhong, S., Miyazaki, S., Aoki, T., and Yasuda, H. Filmmaking production system with rule-based reasoning. In *Image and Vision Computing* (New Zealand, 2003), pp. 366–371.

[141] Johnsen, K., Raij, A., Stevens, A., Lind, D. S., and Lok, B. The validity of a virtual human experience for interpersonal skills education. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, USA, 2007), ACM, pp. 1049–1058.

[142] Jonsson, M., and Hast, A. Animation of water droplet flow on structured surfaces. In *SIGRAD2002* (Linköping, Sweden, 2002), Linköping University Press, pp. 17–22.

[143] Jung, B., Amor, H. B., Heumer, G., and Weber, M. From motion capture to action capture: a review of imitation learning techniques and their application to vr-based character animation. In *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology* (New York, USA, 2006), ACM, pp. 145–154.

[144] Jung, B., and Kopp, S. Flurmax: An interactive virtual agent for entertaining visitors in a hallway. In *IVA* (2003), pp. 23–26.

[145] Jung, Y. Animating and rendering virtual humans - Extending X3D for real-time rendering and animation of virtual characters. In *INSTICC: GRAPP 2008: Proceedings of the Third International Conference on Computer Graphics Theory and Applications* (Setubal, 2008), INSTICC Press, pp. 387–394.

[146] Jung, Y. Building blocks for virtual learning environments. In *Eurographics: WSCG 2008. Communications Papers* (Plzen, 2008), S. Cunningham and V. Skala, Eds., Eurographics Association, pp. 137–143.

[147] Jung, Y., and Behr, J. Extending H-Anim and X3D for advanced animation control. In *Proceedings Web3D 2008* (New York, USA, 2008), S. Spencer, Ed., ACM, pp. 57–65.

[148] Jung, Y., and Behr, J. GPU-based real-time on-surface droplet flow in X3D. In *Web3D 2009* (New York, USA, 2009), S. Spencer, Ed., ACM, pp. 51–54.

[149] Jung, Y., and Behr, J. Simplifying the integration of virtual humans into dialog-like VR systems. In *Proc. of IEEE Virtual Reality 2009 Workshop: 2nd Workshop on Software Engineering and Architectures for Realtime Interactive Systems* (2009), Shaker, pp. 41–50.

[150] Jung, Y., and Behr, J. Towards a new camera model for x3d. In *Proceedings Web3D 2009* (New York, USA, 2009), S. Spencer, Ed., ACM Press, pp. 79–82.

[151] Jung, Y., Franke, T., Dähne, P., and Behr, J. Enhancing X3D for advanced MR appliances. In *Proceedings Web3D '07* (NY, USA, 2007), ACM Press, pp. 27–36.

[152] Jung, Y., Keil, J., Behr, J., Webel, S., Zöllner, M., Engelke, T., Wuest, H., and Becker, M. Adapting X3D for multi-touch environments. In *Proceedings Web3D 2008: 13th International Conference on 3D Web Technology* (New York, USA, 2008), S. Spencer, Ed., ACM Press, pp. 27–30.

[153] Jung, Y., Keil, J., Wuest, H., Engelke, T., Riess, P., and Behr, J. Knowledge at your fingertips - Multi-touch interaction for GIS and architectural design review applications. In *GRAPP 2009: Proc. of the 4th Int. Conf. on CG Theory and Applications* (2009), INSTICC Press, pp. 387–392.

[154] Jung, Y., and Knöpfle, C. Styling and real-time simulation of human hair. In *Intelligent Technologies for Interactive Entertainment. Proceedings: First International Conference, INTETAIN 2005* (Heidelberg, 2005), M. Maybury, O. Stock, and W. Wahlster, Eds., Springer, pp. 240–245.

[155] Jung, Y., and Knöpfle, C. Dynamic aspects of real-time face-rendering. In *Proceedings VRST Cyprus 2006* (New York, 2006), ACM, pp. 193–196.

[156] Jung, Y., and Knöpfle, C. Real-time rendering and animation of virtual characters. *The International Journal of Virtual Reality (IJVR) 6*, 4 (2007), 55–66.

[157] Jung, Y., Recker, R., Olbrich, M., and Bockholt, U. Using X3D for medical training simulations. In *Proceedings Web3D 2008: 13th International Conference on 3D Web Technology* (New York, USA, 2008), S. Spencer, Ed., ACM Press, pp. 43–51.

[158] Jung, Y., Rettig, A., Klar, O., and Lehr, T. Realistic real-time hair simulation and rendering. In *Vision, Video, and Graphics. Proceedings 2005* (Aire-la-Ville, 2005), E. Trucco and M. Chantler, Eds., Eurographics Association, pp. 229–236.

[159] Jung, Y., and Wagner, S. Emotional factors in face rendering. In *IADIS Multi Conference on Computer Science and Information Systems 2010: Proceedings IADIS Interfaces and Human Computer Interaction* (2010), IADIS Press. 5 p.

[160] Jung, Y., Wagner, S., Behr, J., Jung, C., and Fellner, D. W. Storyboarding and pre-visualization with x3d. In *Proceedings Web3D 2010: 15th International Conference on 3D Web Technology* (New York, USA, 2010), S. Spencer, Ed., ACM Press, pp. 73–81.

[161] Jung, Y., Webel, S., Olbrich, M., Drevensek, T., Franke, T., Roth, M., and Fellner, D. W. Interactive textures as spatial user interfaces in x3d. In *Proceedings Web3D 2010: 15th International Conference on 3D Web Technology* (New York, USA, 2010), S. Spencer, Ed., ACM Press, pp. 147–150.

[162] Jung, Y., Weber, C., Keil, J., and Franke, T. Real-time rendering of skin changes caused by emotions. In *Intelligent Virtual Agents: 9th International Conference, IVA 2009* (Heidelberg, 2009), Springer, pp. 504–505.

[163] Kahn, S., Wuest, H., and Fellner, D. W. Time-of-flight based scene reconstruction with a mesh processing tool for model based camera tracking. In *VISAPP 2010* (2010), INSTICC Press, pp. 302–309.

[164] Kajiya, J. T. The rendering equation. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 143–150.

[165] Kajiya, J. T., and Kay, T. L. Rendering fur with three dimensional textures. In *SIGGRAPH '89* (1989), ACM Press, pp. 271–280.

[166] Kalra, P., and Magnenat-Thalmann, N. Modeling of vascular expressions. In *Computer Animation '94* (Geneva, 1994), pp. 50–58.

[167] Kamburelis, M. Shadow maps and projective texturing in x3d. In *Proceedings Web3D '10* (New York, NY, USA, 2010), ACM, pp. 17–26.

[168] Kaneda, K., Ikeda, S., and Yamashita, H. Animation of water droplets moving down a surface. *The Journal of Visualization and Computer Animation 10*, 1 (1999), 15–26.

[169] Kaneda, K., Kagawa, T., and Yamashita, H. Animation of water droplets on a glass plate. *Proceedings of Computer Animation 93* (1993), 177–189.

[170] Kaneda, K., Zuyama, Y., Yamashita, H., and Nishita, T. Animation of water droplet flow on curved surfaces. *Proceedings of Pacific Graphics 96* (1996), 50–65.

[171] Kardan, K., and Casanova, H. Virtual cinematography of group scenes using hierarchical lines of actions. In *Sandbox '08: Proc. of symp. on Video games* (NY, USA, 2008), ACM, pp. 171–178.

[172] Kautz, J. Gpu gems 2. Addison Wesley, 2005, ch. Approximate Bidirectional Texture Functions, pp. 177–187.

[173] Kautz, J., Daubert, K., and Seidel, H.-P. Advanced environment mapping in vr applications. *Computers & Graphics 28*, 1 (2004), 99–104.

[174] Khronos. *OpenGL Specifications*, 2010. http://www.opengl.org/documentation/.

[175] Khronos. Webgl specification, 2010. https://cvs.khronos.org/svn/repos/registry/trunk/public/webgl/doc/spec/WebGL-spec.html.

[176] Kim, J., Park, J., Kim, H., and Lee, C. Hci (human computer interaction) using multi-touch tabletop display. *Communications, Computers and Signal Processing, 2007. PacRim 2007.* (2007), 391–394.

[177] Kim, T.-Y., and Neumann, U. Opacity shadow maps. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (London, UK, 2001), Springer-Verlag, pp. 177–182.

[178] Kim, T.-Y., and Neumann, U. Interactive multiresolution hair modeling and editing. In *SIGGRAPH '02* (2002), ACM Press, pp. 620–629.

[179] King, G. Real-time computation of dynamic irradiance environment maps. In *GPU Gems 2*, M. Pharr, Ed. Addison Wesley, 2005, ch. 10, pp. 167–176.

[180] Kipp, M., Heloir, A., Gebhard, P., and Schröder, M. Realizing multimodal behavior: Closing the gap between behavior planning and embodied agent presentation. In *Intelligent Virtual Agents: 10th Intl. Conf., IVA 2010* (2010), Springer.

[181] Klesen, M., and Gebhard, P. Affective multimodal control of virtual characters. *IJVR 6*, 4 (2007), 43–54.

[182] Kneafsey, J., and McCabe, H. Camera control through cinematography for virtual environments: A state of the art report. In *Proc. of Eurographics Ireland Chapter Workshop 2004* (2004).

[183] Knöpfle, C., and Jung, Y. The virtual human platform: Simplifying the use of virtual characters. *The International Journal of Virtual Reality (IJVR) 5*, 2 (2006), 25–30.

[184] Koenderink, J., and Pont, S. The secret of velvety skin. *Machine Vision and Applications 14*, 4 (2003), 260–268.

[185] Koh, C. K., and Huang, Z. A simple physics model to animate human hair modeled in 2d strips in real time. In *Proc. of the EG workshop on Computer animation and simulation* (2001), Springer-Verlag New York, Inc., pp. 127–138.

[186] Kohlgrueber, K. *Der gleichläufige Doppelschneckenextruder: Grundlagen, Technologie, Anwendungen.* Hanser, 2008.

[187] Kölzer, K., Jung, Y., Nagl, F., Birnbach, B., and Grimm, P. Grafikkartenbasierte Simulation von tröpfchenförmigen Flüssigkeiten auf Oberflächen. In *5. Workshop der GI-Fachgruppe VR/AR* (Aachen, 2008), Shaker, pp. 149–156.

[188] Kong, D., Lao, W., and Yin, B. An improved algorithm for hairstyle dynamics. In *ICMI '02: Proceedings of the 4th IEEE International Conference on Multimodal Interfaces* (2002), IEEE Computer Society, p. 535.

[189] Kopp, S., Krenn, B., Marsella, S., Marshall, A., Pelachaud, C., Pirker, H., Thórisson, K., and Vilhjálmsson, H. Towards a common framework for multimodal generation: The behavior markup language. In *IVA* (2006), pp. 205–217.

[190] Korn, M., Stange, M., von Arb, A., Blum, L., Kreil, M., Kunze, K.-J., Anhenn, J., Wallrath, T., and Grosch, T. Interactive augmentation of live images using a hdr stereo camera.

[191] Koster, M., Haber, J., and Seidel, H.-P. Real-time rendering of human hair using programmable graphics hardware. In *CGI '04: Proceedings of the Computer Graphics International* (2004), IEEE Computer Society, pp. 248–256.

[192] Kovar, L., Gleicher, M., and Pighin, F. Motion graphs. *ACM Trans. Graph. 21*, 3 (2002), 473–482.

[193] Kranstedt, A., Kopp, S., and Wachsmuth, I. Murml: A multimodal utterance representation markup language for conversational agents. In *Embodied conversational agents - let's specify and evaluate them! Proc. of AAMAS Workshop* (2002).

[194] Krenn, B., and Pirker, H. Defining the gesticon: Language and gesture coordination for interacting embodied agents. In *Proc. AISB'04 Symp. on Language, Speech and Gesture for Expressive Characters* (University of Leeds, UK, 2004), pp. 107–115.

[195] Krishnaswamy, A., and Baranoski, G. V. G. A biophysically-based spectral model of light interaction with human skin. *Computer Graphics Forum 23*, 3 (2004), 331–340.

[196] Kshirsagar, S., Magnenat-Thalmann, N., Guye-Vuillème, A., Thalmann, D., Kamyab, K., and Mamdani, E. Avatar markup language. In *EGVE '02: Proceedings of the workshop on Virtual environments 2002* (Aire-la-Ville, Switzerland, 2002), Eurographics Association, pp. 169–177.

[197] Küblbeck, C. Shore, 2010. www.iis.fhg.de/EN/bf/bv/kognitiv/biom/dd.jsp.

[198] Kubo, H., Dobashi, Y., and Morishima, S. Curvature-dependent reflectance function for rendering translucent materials. In *SIGGRAPH '10: ACM SIGGRAPH 2010 Talks* (New York, NY, USA, 2010), ACM, pp. 1–1.

[199] Křivánek, J., Konttinen, J., Pattanaik, S., Bouatouch, K., and Žára, J. Fast approximation to spherical harmonics rotation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches* (New York, USA, 2006), ACM, p. 154.

[200] LeBlanc, A. M., Turner, R., and Thalmann, D. Rendering hair using pixel blending and shadow buffers. *The Journal of Visualization and Computer Animation 2*, 3 (1991), 92–97.

[201] Lee, J., Chai, J., Reitsma, P. S. A., Hodgins, J. K., and Pollard, N. S. Interactive control of avatars animated with human motion data. *ACM Trans. Graph. 21*, 3 (2002), 491–500.

[202] Lindner, H. *Physik für Ingenieure*, 15 ed. Fachbuchverlag Leipzig, München, 1999.

[203] Liu, Y., Zhu, H., Liu, X., and Wu, E. Real-time simulation of physically based on-surface flow. *The Visual Computer 21*, 8 (2005), 727–734.

[204] Löckelt, M., Pfleger, N., and Reithinger, N. Multi-party conversation for mixed reality. *IJVR 6*, 4 (2007), 31–42.

[205] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87* (1987), pp. 163–169.

[206] Loscos, C., Frasson, M.-C., Drettakis, G., Walter, B., Granier, X., and Poulin, P. Interactive virtual relighting and remodeling of real scenes. In *Rendering techniques (Proc. 10th EG Workshop on Rend.)* (1999), pp. 235–246.

[207] MacDorman, K. F. Androids as an experimental apparatus: Why is there an uncanny valley and can we exploit it? *Proceedings of the 2005 CogSci Workshop: Toward Social Mechanisms of Android Science* (2005), 106–118.

[208] MacDorman, K. F., Green, R. D., Ho, C.-C., and Koch, C. T. Too real for comfort? uncanny responses to computer generated faces. *Computers in Human Behavior 25*, 3 (2009), 695–710.

[209] Madsen, C. B., and Laursen, R. E. A scalable gpu-based approach to shading and shadowing for photorealistic real-time augmented reality. In *GRAPP 2007: Proceedings* (2007), INSTICC Press, pp. 252–261.

[210] Magnenat-Thalmann, N., and Egges, A. Interactive virtual humans in real-time virtual environments. *IJVR 5*, 2 (2006), 15–24.

[211] Mariauzouls, C. *Psychophysiologie von Scham und Erröten*. Dissertation, Ludwig-Maximilians-Universität München, 1996.

[212] Marriott, A. VHML, 2001. http://www.vhml.org/.

[213] Marschner, S. R., Jensen, H. W., Cammarano, M., Worley, S., and Hanrahan, P. Light scattering from human hair fibers. *ACM Trans. Graph. 22*, 3 (2003), 780–791.

[214] Marsella, S., Gratch, J., and Petta, P. Computational models of emotion. In *A blueprint for an affectively competent agent: Cross-fertilization between Emotion Psychology, Affective Neuroscience, and Affective Computing*, K. R. Scherer, T. Bänziger, and E. Roesch, Eds., Oxford University Press. In press.

[215] Martin, T., and Tan, T.-S. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Eurographics Symposium on Rendering (2004)* (2004), Eurographics Association, pp. 153–160.

[216] Maxon. Cinema 4d, 2010. http://www.maxon.net/products/cinema-4d.html.

[217] McCann, J., and Pollard, N. Responsive characters from motion fragments. *ACM Trans. Graph. 26*, 3 (2007), 6.

[218] Menzel, N., and Guthe, M. g-brdfs: An intuitive and editable btf representation. *Computer Graphics Forum 28*, 8 (2009), 2189–2200.

[219] Mertens, T., Kautz, J., Bekaert, P., Reeth, F. V., and Seidel, H.-P. Efficient rendering of local subsurface scattering. In *PG '03: Proc. of the 11th Pacific Conf. on CG and Applications* (Washington, DC, USA, 2003), IEEE Computer Society, pp. 51–58.

[220] Mittring, M. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), ACM, pp. 97–121.

[221] Mori, M. The uncanny valley. *Energy 7*, 4 (1970), 33–35.

[222] Multon, F., France, L., Cani-Gascuel, M.-P., and Debunne, G. Computer animation of human walking: a survey. *The Journal of Visualization and Computer Animation 10*, 1 (1999), 39–54.

[223] Ng, R., Ramamoorthi, R., and Hanrahan, P. Triple product wavelet integrals for all-frequency relighting. *ACM Trans. Graph. 23*, 3 (2004), 477–487.

[224] Nguyen, H., and Donnelly, W. Hair animation and rendering in the nalu demo. In *GPU Gems 2*. Addison Wesley, 2005, ch. 23, pp. 361–380.

[225] Niewiadomski, R., Bevacqua, E., Mancini, M., and Pelachaud, C. Greta: an interactive expressive eca system. In *AAMAS (2)* (2009), pp. 1399–1400.

[226] N.Magnenat-Thalmann, S.Hadap, and P.Kalra. State of the art in hair simulation. In *Int. Workshop on Human Modeling and Animation, Korea Computer Graphics Society* (2002), pp. 3–9.

[227] NVIDIA. *CUDA Compute Unified Device Architecture - Programming Guide*, 6 2007. Version 1.0.

[228] NVIDIA. SceniX, 2010. http://www.nvidia.com/object/scenix.html.

[229] Octaga. Octaga – bringing enterprise data to life, 2009. http://www.octaga.com/.

[230] OpenCV. Opencv wiki, 2009. http://opencv.willowgarage.com/wiki/.

[231] Open scene graph, 2009. http://www.openscenegraph.org.

[232] OpenSG. OpenSG, 2010. http://www.opensg.org/.

[233] Ortony, A., Clore, G., and Collins, A. *The Cognitive Structure of Emotions*. Cambridge University Press, 1988.

[234] O'Toole, A. J., Price, T., Vetter, T., Bartlett, J. C., and Blanz, V. 3d shape and 2d surface textures of human faces: the role of "averages" in attractiveness and age. *Image and Vision Computing 18*, 1 (1999), 9 – 19.

[235] Owens, J. Gpu gems 2. Addison Wesley, 2005, ch. Streaming Architectures and Technology Trends, pp. 457–470.

[236] Oyarzun, D., Lehr, M., Ortiz, A., del Puy Carretero, M., Ugarte, A., Vivanco, K., and García-Alonso, A. Using virtual characters as tv presenters. In *Technologies for E-Learning and Digital Entertainment, Edutainment 2007* (2007), LNCS, Springer, pp. 225–236.

[237] Oyarzun, D., Ortiz, A., del Puy Carretero, M., Gelissen, J., Garcia-Alonso, A., and Sivan, Y. Adml: a framework for representing inhabitants in 3d virtual worlds. In *Proceedings Web3D '09* (New York, USA, 2009), ACM, pp. 83–90.

[238] Pan, X., Gillies, M., and Slater, M. The impact of avatar blushing on the duration of interaction between a real and virtual person. In *Presence 2008: The 11th Annual International Workshop on Presence* (2008), pp. 100–106.

[239] Pandzic, I., and Forchheimer, R. *MPEG-4 Facial Animation: The Standard, Implementation and Applications*. John Wiley & Sons, West Sussex, England, 2002.

[240] Papagiannakis, G. *An illumination registration model for dynamic virtual humans in mixed reality*. Dissertation, University of Geneva, 2006.

[241] Park, S. I., Shin, H. J., Kim, T. H., and Shin, S. Y. On-line motion blending for real-time locomotion generation. *Comput. Animat. and Virtual Worlds 15*, 3-4 (2004), 125–138.

[242] Park, S. I., Shin, H. J., and Shin, S. Y. On-line locomotion generation based on motion blending. In *Proceedings of the 2002 ACM SIGGRAPH/ Eurographics Symposium on Computer Animation* (2002), pp. 105–111.

[243] Patel, M. Colouration issues in computer generated facial animation. *Computer Graphics Forum 14*, 2 (1995), 117–126.

[244] Pettré, J., Kallmann, M., and Lin, M. C. Motion planning and autonomy for virtual humans. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes* (New York, USA, 2008), ACM, pp. 1–31.

[245] Pharr, M., and Green, S. Ambient occlusion. In *GPU Gems*. Addison Wesley, 2004, ch. 17, pp. 279–292.

[246] Pilet, J., Geiger, A., Lagger, P., Lepetit, V., and Fua, P. An all-in-one solution to geometric and photometric calibration. In *Int. Symposium on Mixed and Augmented Reality* (Santa Barbara, CA, 2006).

[247] Piwek, P., Krenn, B., Schröder, M., Grice, M., Baumann, S., and Pirker, H. Rrl: A rich representation language for the description of agent behaviour in neca. In *Embodied conversational agents - let's specify and evaluate them! Proc. of AAMAS Workshop* (2002).

[248] Platt, S. M., and Badler, N. I. Animating facial expressions. *SIGGRAPH '81 15*, 3 (1981), 245–252.

[249] Plutchik, R. *Emotion: Theory, Research and Experience*. Academic Press, Sheffield, 1980.

[250] Pomi, A., and Slusallek, P. Interactive Mixed Reality Rendering in a Distributed Ray Tracing Framework. In *IEEE and ACM Int. Symp. on Mixed a. Augmented Reality ISMAR 2004* (2004).

[251] Ponder, M., Papagiannakis, G., Molet, T., Magnenat-Thalmann, N., and Thalmann, D. VHD++ development framework: Towards extendible, component based VR/AR simulation engine featuring advanced virtual character technologies. *CGI '03: Proceedings of Computer Graphics International* (2003), 96–106.

[252] Prandtl, L., Oertel, H., and Böhle, M. *Prandtl-Führer durch die Strömungslehre*, 10 ed. Vieweg, 2001.

[253] Preda, M., and Preteux, F. Virtual character within mpeg-4 animation framework extension. *IEEE Transactions on Circuits and Systems for Video Technology 14*, 7 (2004), 975–988.

[254] Prendinger, H., and Ishizuka, M. *Life-Like Characters*. Springer, Heidelberg, 2004.

[255] Press, W. H., Vetterling, W. T., Teukolsky, S. A., and Flannery, B. P. *Numerical Recipes in C++: The Art of Scientific Computing*, second ed. Cambridge University Press, 2002.

[256] Ramamoorthi, R., and Hanrahan, P. An efficient representation for irradiance environment maps. In *Proceedings of ACM SIGGRAPH 2001* (2001), Computer Graphics Proceedings, Annual Conference Series, pp. 497–500.

[257] Reeves, W. T., Salesin, D. H., and Cook, R. L. Rendering antialiased shadows with depth maps. In *SIGGRAPH '87* (1987), ACM Press, pp. 283–291.

[258] Reiners, D. *OpenSG: A Scene Graph System for Flexible and Efficient Realtime Rendering for Virtual and Augmented Reality Applications*. PhD thesis, Technische Universität Darmstadt, 2002.

[259] Reynolds, C. Opensteer, 2005. http://opensteer.sf.net/.

[260] Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87* (New York, NY, USA, 1987), ACM, pp. 25–34.

[261] Reynolds, C. W. Steering behaviors for autonomous characters. In *Game Developers Conference 1999* (1999), pp. 763–782.

[262] Riess, P., and Jung, Y. Module- and chain-based architecture for creating virtual and augmented reality manuals. In *Schumann, Marco (Hrsg.) u.a.; 5. Workshop der GI-Fachgruppe VR/AR* (Aachen, 2008), Shaker, pp. 197–208.

[263] Roewen, J., and Glek, T. J-alice, 2003. http://j-alice.sourceforge.net/.

[264] Rosado, G. Motion blur as a post-processing effect. In *GPU Gems 3*. Addison-Wesley, 2007, ch. 27, pp. 575–581.

[265] Rose, C., Choen, M., and Bodenheimer, P. Verbs and adverbs. multidimensional motion interpolation. In *IEEE CG and Applications* (1998).

[266] Rost, R. J. *OpenGL Shading Language*, 2 ed. Addison-Wesley, Boston, 2006.

[267] Roussou, M. Immersive interactive virtual reality and informal education. In *Proceedings of User Interfaces for All: Interactive Learning Environments for Children* (Athens, 2000).

[268] Safanova, A., and Hodgins, J. K. Synthesizing human motion from intuitive constraints. In *Artificial Intelligence Techniques for Computer Graphics*. Springer-Verlag, 2009, ch. 2, pp. 15–39.

[269] Sander, P. V., Gosselin, D., and Mitchell, J. L. Real-time skin rendering on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches* (New York, NY, USA, 2004), ACM, p. 148.

[270] Sannier, G., Balcisoy, S., Magnenat-Thalmann, N., and Thalmann, D. Vhd: A system for directing real-time virtual actors. *The Visual Computer 15* (1999), 320–329.

[271] Sattler, M., Sarlette, R., Zachmann, G., and Klein, R. Hardware-accelerated ambient occlusion computation. In *Vision, Modeling, and Visualization 2004* (2004), B. Girod, M. Magnor, and H.-P. Seidel, Eds., pp. 331–338.

[272] Schachter, S., and Singer, J. E. Cognitive, social, and physiological determinants of emotional state. *Psychological Review*, 69 (1962), 379–399.

[273] Schandry, R. *Psychophysiologie: Körperliche Indikatoren menschlichen Verhaltens*. Urban & Schwarzenberg, München, Germany, 1981.

[274] Scherbaum, K., Sunkel, M., Seidel, H.-P., and Blanz, V. Prediction of individual non-linear aging trajectories of faces. In *The European Association for Computer Graphics, EUROGRAPHICS 2007* (Prague, Czech Republic, 2007), vol. 26 of *Computer Graphics Forum*, Blackwell, pp. 285–294.

[275] Scherzer, D., Wimmer, M., and Purgathofer, W. A survey of real-time hard shadow mapping methods. In *EG 2010 - State of the Art Reports* (2010), H. Hauser and E. Reinhard, Eds., Eurographics Association, pp. 21–36.

[276] Scheuermann, T. Practical real-time hair rendering and shading. Siggraph04 Sketches, 2004.

[277] Scheuermann, T., and Tatarchuk, N. Advanced depth of field rendering. In *ShaderX3: Advanced Rendering Techniques in DirectX and OpenGL*, W. Engel, Ed. Charles River Media, Cambridge, 2004.

[278] Schlick, C. An inexpensive brdf model for physically based rendering. 149–162. Eurographics 94, Computer Graphics Forum 13(3).

[279] Schwenk, K., Jung, Y., Behr, J., and Fellner, D. W. A modern declarative surface shader for x3d. In *Proceedings Web3D 2010: 15th Int. Conference on 3D Web Technology* (New York, USA, 2010), S. Spencer, Ed., ACM Press, pp. 7–15.

[280] Sederberg, T. W., and Parry, S. R. Free-form deformation of solid geometric models. In *SIGGRAPH '86* (New York, USA, 1986), ACM, pp. 151–160.

[281] Shapiro, A., Chu, D., Allen, B., and Faloutsos, P. A dynamic controller toolkit. In *Sandbox '07: Proceedings of the 2007 ACM SIGGRAPH symposium on Video games* (New York, USA, 2007), ACM, pp. 15–20.

[282] Shapiro, A., Pighin, F., and Faloutsos, P. Hybrid control for interactive character animation. In *PG '03: Proc. of the 11th Pacific Conf. on CG and Applications* (2003), IEEE, pp. 455–461.

[283] Shastry, A. S. Soft-edged shadows, 2005. http://www.gamedev.net/reference/articles/article2193.asp.

[284] Shearn, D., Bergman, E., Hill, K., Abel, A., and Hinds, L. Facial coloration and temperature responses in blushing. *Psychophysiology 27*, 6 (1990), 687–693.

[285] Shirley, P. *Realistic Ray Tracing*. AK Peters, 2000.

[286] Shreiner, D., Woo, M., Neider, J., and Davis, T. *OpenGL Programming Guide*, 5 ed. Addison-Wesley, Boston, 2006.

[287] Sillion, F. X., and Puech, C. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[288] Sloan, P.-P., Kautz, J., and Snyder, J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics 21*, 3 (2002), 527–536.

[289] Sloan, P.-P., Luna, B., and Snyder, J. Local, deformable precomputed radiance transfer. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM Press, pp. 1216–1224.

[290] Sloan, P.-P. J., Charles F. Rose, I., and Cohen, M. F. Shape by example. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (NY, USA, 2001), ACM, pp. 135–143.

[291] Sobotta, J., and Becher, H. *Atlas der Anatomie des Menschen*, 17. ed. Band 3: Zentralnervensystem, Autonomes Nervensystem, Sinnesorgane und Haut, Periphere Leitungsbahnen. Urban & Schwarzenberg, München, Germany, 1973.

[292] Sobottka, G., and Weber, A. Übersicht über die optischen Eigenschaften von Human-Haar unter besonderer Berücksichtigung der Anforderungen der Computergraphik. Tech. Rep. CG-2002-1, Universität Bonn, 2002.

[293] Sobottka, G., and Weber, A. Geometrische und Physikalische Eigenschaften von Humanhaar. Tech. Rep. CG-2003-1, Universität Bonn, 2003.

[294] Sousa, T. Gpu gems 2. Addison Wesley, 2005, ch. Generic Refraction Simulation, pp. 295–305.

[295] Stamminger, M., and Drettakis, G. Perspective shadow maps. In *SIGGRAPH '02* (New York, NY, USA, 2002), ACM, pp. 557–562.

[296] Starck, J., and Hilton, A. Surface capture for performance-based animation. *IEEE Comput. Graph. Appl. 27*, 3 (2007), 21–31.

[297] Strauss, P., and Carey, R. An object-oriented 3D graphics toolkit. *ACM Computer Graphics* (1992).

[298] Stricker, D., Zoellner, M., Bisler, A., and Lutz, B. Traveling in time and space with virtual and augmented reality. In *ETH Zürich: Recording, Modeling and Visualization of Cultural Heritage* (London, 2006), E. Baltsavias, Ed., pp. 431–439.

[299] Su, Y., Wang, W., Xu, K., and Jiang, C. The optical properties of skin. 299–304. Optics in Health Care and Biomedical Optics: Diagnostics and Treatment 4916.

[300] Sudarsky, S. Generating dynamic shadows for virtual reality applications. In *IV '01: Proceedings of the Fifth International Conference on Information Visualisation* (Washington, DC, USA, 2001), IEEE Computer Society, p. 595.

[301] Sun. Java3d, 2008. https://java3d.dev.java.net/.

[302] Sung, M., Kovar, L., and Gleicher, M. Fast and accurate goal-directed motion synthesis for crowds. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/ Eurographics symposium on Computer animation* (New York, USA, 2005), ACM, pp. 291–300.

[303] Supan, P., and Stuppacher, I. Interactive image based lighting in augmented reality. In *Central European Seminar on Computer Graphics* (2006).

[304] Swarz, J., Ousley, A., Magro, A., Rienzo, M., Burns, D., Lindsey, A. M., Wilburn, B., and Bolcar, S. Cancerspace: A simulation-based game for improving cancer-screening rates. *IEEE Computer Graphics & Applications 30*, 1 (2010), 90–94.

[305] Szirmay-Kalos, L., Szecsi, L., and Sbert, M. GPUGI: Global illumination effects on the gpu. In *Eurographics 2006: Tutorials* (2006), Eurographics Association, pp. 1201–1278.

[306] Tariq, S., and Bavoil, L. Real time hair simulation and rendering on the gpu. Siggraph '08 Sketches, 2008.

[307] Tatarchuk, N. Artist-directable real-time rain rendering in city environments. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses* (New York, USA, 2006), ACM, pp. 23–64.

[308] Tchou, C., and Debevec, P. Hdr shop, 2001. http://www.hdrshop.com/.

[309] Thalmann, D., and Musse, S. R. *Crowd Simulation*. Springer, London, 2007.

[310] Thiebaux, M., Marshall, A., Marsella, S., and Kallman, M. Smartbody: Behavior realization for embodied conversational agents. In *Proc. of the Intl. Conf. on Autonomous Agents and Multiagent Systems* (2008).

[311] Tolani, D., Goswami, A., and Badler, N. I. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graph. Models Image Process. 62*, 5 (2000), 353–388.

[312] Tramberend, H. Avocado – a distributed virtual environment framework. In *Proceedings of IEEE Virtual Reality 1999* (Houston, Texas, 1999), pp. 14–21.

[313] Tümmler, J. *Avatare in Echtzeitsimulationen*. Dissertation, Universität Kassel, 2007.

[314] Ulich, D., and Mayring, P. *Psychologie der Emotionen*. Kohlhammer, Stuttgart, Germany, 2003.

[315] van Tol, W., and Egges, A. 3d characters that are moved to tears. In *Short Paper and Poster Proceedings Computer Animation and Social Agents Conference* (Amsterdam, 2009).

[316] van Tol, W., and Egges, A. Real-time crying simulation. In *Intelligent Virtual Agents: 9th International Conference, IVA 2009* (Heidelberg, 2009), Springer, pp. 215–228.

[317] van Welbergen, H., van Basten, B. J. H., Egges, A., Ruttkay, Z., and Overmars, M. H. Real Time Animation of Virtual Humans: A Trade-off Between Naturalness and Control. Eurographics Association, pp. 45–72.

[318] Vilhjálmsson, H., Cantelmo, N., Cassell, J., E. Chafai, N., Kipp, M., Kopp, S., Mancini, M., Marsella, S., Marshall, A. N., Pelachaud, C., Ruttkay, Z., Thórisson, K. R., Welbergen, H., and Werf, R. J. The behavior markup language: Recent developments and challenges. In *IVA '07: Proceedings of the 7th international conference on Intelligent Virtual Agents* (Berlin, Heidelberg, 2007), Springer-Verlag, pp. 99–111.

[319] Vinayagamoorthy, V., Gillies, M., Steed, A., Tanguy, E., Pan, X., Loscos, C., and Slater, M. Building Expression into Virtual Characters . In *Eurographics 2006 - State of the Art Reports* (2006), Eurographics Association, pp. 21–61.

[320] Vingerhoets, A. J. J. M., and Cornelius, R. R. *Adult Crying: A Biopsychosocial Approach*. Brunner-Routledge, San Francisco, 2001.

[321] Virtual human, 2006. http://www.virtual-human.de/.

[322] Volino, P., and Magnenat-Thalmann, N. Animating complex hairstyles in

real-time. In *VRST '04: Proc. of the ACM symposium on VR software a. technology* (NY, USA, 2004), ACM, pp. 41–48.

[323] W3C. Synchronized multimedia integration language (smil 2.0) - 2nd ed., 2005. http://www.w3.org/TR/2005/REC-SMIL2-20050107/.

[324] W3C. Svg filters 1.2, part 2: Language (working draft), 2007. http://www.w3.org/TR/2007/WD-SVGFilter12-20070501/.

[325] Wächter, C., and Keller, A. Instant ray tracing: The bounding interval hierarchy. *Proceedings of the 17th Eurographics Symposium on Rendering* (2006), 139–149.

[326] Wahlster, W. Dialogue systems go multimodal: The smartkom experience. In *SmartKom: Foundations of Multimodal Dialogue Systems*. Springer, Heidelberg, 2006, pp. 3–27.

[327] Walczak, K., Wojciechowski, R., and Cellary, W. Dynamic interactive vr network services for education. In *VRST Cyprus 2006. Proceedings* (New York, USA, 2006), ACM, pp. 277–286.

[328] Wall, M. Galib: A c++ library of genetic algorithm components, 2006. http://lancet.mit.edu/ga/.

[329] Wang, H., Mucha, P. J., and Turk, G. Water drops on surfaces. *ACM Trans. Graph. 24*, 3 (2005), 921–929.

[330] Ward, K., Bertails, F., Kim, T.-Y., Marschner, S. R., Cani, M.-P., and Lin, M. C. A survey on hair modeling: Styling, simulation, and rendering. *IEEE Trans. Vis. Comput. Graph. 13*, 2 (2007), 213–234.

[331] Ward, K., Galoppo, N., and Lin, M. Interactive virtual hair salon. *Presence: Teleoper. Virt. Environ. 16*, 3 (2007), 237–251.

[332] Ward, K., and Lin, M. C. Adaptive grouping and subdivision for simulating hair dynamics. In *PG '03: Proc. of the 11th Pacific Conference on CG and Applications* (2003), IEEE, pp. 234 – 242.

[333] Ward, K., Lin, M. C., Lee, J., Fisher, S., and Macri, D. Modeling hair using level-of-detail representations. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)* (2003), IEEE Computer Society, pp. 41 – 49.

[334] Watzlawick, P., Beavin, J., and Jackson, D. *Menschliche Kommunikation. Formen, Störungen, Paradoxien*. Huber Verlag, Bern, 1969.

[335] Web3DConsortium. H-anim, 2005. http://www.web3d.org/x3d/specifications/ISO-IEC-19774-HumanoidAnimation/.

[336] Web3DConsortium. X3D, 2008. http://www.web3d.org/x3d/specifications/.

[337] Web3DConsortium. SAI, 2009. http://www.web3d.org/x3d/specifications/ISO-IEC-FDIS-19775-2.2-X3D-SceneAccessInterface/.

[338] Weber, C., and Jung, Y. Entwurf und Evaluation eines Modells zur Darstellung emotional verursachter Hautveränderungen. In *Gerndt, Andreas (Hrsg.) u.a.; 6. Workshop der GI-Fachgruppe VR/AR* (Aachen, 2009), Shaker, pp. 245–256.

[339] Weber, J. Run-time skin deformation. In *Proceedings of Game Developers Conference* (2000).

[340] Weekley, J. D., Blais, C. L., and Brutzman, D. Composing behaviors and swapping bodies with motion capture data in x3d. In *Web3D '07: Proc. of the twelfth int. conference on 3D web technology* (NY, USA, 2007), ACM, pp. 195–200.

[341] Weekley, J. D., and Brutzman, D. P. Beyond viewpoint: X3d camera nodes for digital cinematography. In *Web3D '09* (NY, USA, 2009), ACM, pp. 71–77.

[342] Weizenbaum, J. Eliza — a computer program for the study of natural language communication between man and machine. *Commun. ACM 9*, 1 (1966), 36–45.

[343] Wikipedia. Dialog system — wikipedia, the free encyclopedia, 2010. http://en.wikipedia.org/wiki/Dialog_system.

[344] Williams, L. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (1978), 270–274.

[345] Wimmer, M., Scherzer, D., and Purgathofer, W. Light space perspective shadow maps. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering 2004)* (2004), A. Keller and H. W. Jensen, Eds., pp. 143–151.

[346] Witkin, A., and Baraff, D. Physically based modeling: Principles and practice. In *ACM SIGGRAPH 1997 courses* (1997), ACM Press.

[347] Xj3D. Xj3d dynamic texture rendering extension, 2004. http://www.xj3d.org/extensions/render_texture.html.

[348] Yang, X., Petriu, D., Whalen, T., and Petriu, E. Hierarchical animation control of avatars in 3-d virtual environments. *IEEE Trans. on Inst. and Meas. 54*, 3 (2005), 1333–1341.

[349] Yannopoulos, A., Savrami, K., and Varvarigou, T. Directornotation, a tool for AmI & intelligent content: an introduction by example. *International Journal of Cognitive Informatics and Natural Intelligence (IJCiNi)* (2008).

[350] Yu, Y. Modeling realistic virtual hairstyles. In *Proceedings of Pacific Graphics* (2001), pp. 295–304.

[351] Yuksel, C., and Keyser, J. Deep opacity maps. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008) 27*, 2 (2008), 675–680.

[352] Yuksel, C., and Tariq, S. Advanced techniques in real-time hair rendering and simulation. In *SIGGRAPH '10: ACM SIGGRAPH 2010 Courses* (New York, NY, USA, 2010), ACM, pp. 1–168.

[353] Y.Wu, P.Kalra, and Magnenat-Thalmann, N. Physically-based wrinkle simulation and skin rendering. In *EGCAS97* (1997), pp. 69–79.

[354] Zhang, F., Sun, H., Xu, L., and Lun, L. K. Parallel-split shadow maps for large-scale virtual environments. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications* (New York, NY, USA, 2006), ACM, pp. 311–318.

[355] Zinke, A., Yuksel, C., Weber, A., and Keyser, J. Dual scattering approximation for fast multiple scattering in hair. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008) 27*, 3 (2008), 1–10.

[356] Zoeckler, M., Stalling, D., and Hege, H.-C. Interactive visualization of 3d-vector fields using illuminated streamlines. *Proceedings of IEEE Visualization '96* (1996), 107 – 113.

Publications

[1] Yvonne Jung, Alexander Rettig, Oliver Klar, and Timo Lehr. Realistic real-time hair simulation and rendering. In Emanuele Trucco and Mike Chantler, editors, *Eurographics: Vision, Video, and Graphics. Proceedings 2005*, pages 229–236, Aire-la-Ville, 2005. Eurographics Association.

[2] Yvonne Jung and Christian Knöpfle. Styling and real-time simulation of human hair. In Mark Maybury, Oliviero Stock, and Wolfgang Wahlster, editors, *Intelligent Technologies for Interactive Entertainment. Proceedings: First International Conference, INTETAIN 2005*, volume 3814 of *Lecture Notes in Artificial Intelligence*, pages 240–245, Heidelberg, 2005. Springer.

[3] Christian Knöpfle and Yvonne Jung. The virtual human platform: Simplifying the use of virtual characters. *The International Journal of Virtual Reality (IJVR)*, 5(2):25–30, 2006.

[4] Yvonne Jung and Christian Knöpfle. Dynamic aspects of real-time face-rendering. In *VRST Cyprus 2006. Proceedings of the ACM symposium on Virtual Reality software and technology*, pages 193–196, New York, USA, 2006. ACM.

[5] Johannes Behr, Patrick Dähne, Yvonne Jung, and Sabine Webel. Beyond the web browser – X3D and immersive VR. In *IEEE VR 2007: VR Tutorial and Workshop Proceedings: IEEE Symposium on 3D UI*, Piscataway, USA, 2007. IEEE. 5 p.

[6] Johannes Behr, Patrick Dähne, and Yvonne Jung. Avalon – advanced mixed reality technology at your fingertips. *Computer Graphics topics*, 19(1):31–32, 2007.

[7] Yvonne Jung, Tobias Franke, Patrick Dähne, and Johannes Behr. Enhancing X3D for advanced MR appliances. In *Web3D '07: Proceedings of the twelfth international conference on 3D web technology*, pages 27–36, New York, NY, USA, 2007. ACM Press. (Second Price Best Paper Award INI-GraphicsNet 2008).

[8] Yvonne Jung, Christine Weber, Sabine Webel, and Christian Knöpfle. Exploration einer virtuellen Stratigraphie mit Hilfe von Verfahren des interaktiven Designreviews. In Marc Erich Latoschik and Bernd Fröhlich, editors, *GI-Fachgruppe Virtuelle und Erweiterte Realität: 4. Workshop der GI-Fachgruppe VR/AR*, pages 47–54, Aachen, 2007. Shaker.

[9] Yvonne Jung and Christian Knöpfle. Real-time rendering and animation of virtual characters. *The International Journal of Virtual Reality (IJVR)*, 6(4):55–66, 2007.

[10] Tobias Franke and Yvonne Jung. Real-time mixed reality with GPU techniques. In *INSTICC: GRAPP 2008: Proceedings of the Third International Conference on Computer Graphics Theory and Applications*, pages 249–252, Setubal, 2008. INSTICC Press.

[11] Yvonne Jung. Animating and rendering virtual humans – Extending X3D for real-time rendering and animation of virtual characters. In *INSTICC: GRAPP 2008:*

Proceedings of the Third International Conference on Computer Graphics Theory and Applications, pages 387–394, Setubal, 2008. INSTICC Press.

[12] Yvonne Jung. Building blocks for virtual learning environments. In Steve Cunningham and Vaclav Skala, editors, *Eurographics: WSCG 2008. Communications Papers*, pages 137–143, Plzen, 2008. Eurographics Association.

[13] Ulrich Bockholt, Yvonne Jung, Ruth Recker, and Manuel Olbrich. Visuelles und haptisches Volumenrendering von medizinischen Bilddaten in Virtual Reality-Simulationen. In Michael Schenk, editor, *11. IFF-Wissenschaftstage 2008. Tagungsband*, pages 31–37, Magdeburg, 2008. Fraunhofer IFF.

[14] Yvonne Jung, Ruth Recker, Manuel Olbrich, and Ulrich Bockholt. Using X3D for medical training simulations. In Stephen Spencer, editor, *Proceedings Web3D 2008: 13th International Conference on 3D Web Technology*, pages 43–51, New York, USA, 2008. ACM Press.

[15] Yvonne Jung, Jens Keil, Johannes Behr, Sabine Webel, Michael Zöllner, Timo Engelke, Harald Wuest, and Mario Becker. Adapting X3D for multi-touch environments. In Stephen Spencer, editor, *Proceedings Web3D 2008: 13th International Conference on 3D Web Technology*, pages 27–30, New York, USA, 2008. ACM Press.

[16] Tobias Franke and Yvonne Jung. Precomputed radiance transfer for X3D based mixed reality applications. In Stephen Spencer, editor, *Proceedings Web3D 2008: 13th International Conference on 3D Web Technology*, pages 7–10, New York, USA, 2008. ACM Press.

[17] Yvonne Jung and Johannes Behr. Extending H-Anim and X3D for advanced animation control. In Stephen Spencer, editor, *Proceedings Web3D 2008: 13th International Conference on 3D Web Technology*, pages 57–65, New York, USA, 2008. ACM Press.

[18] Konrad Kölzer, Yvonne Jung, Frank Nagl, Bastian Birnbach, and Paul Grimm. Grafikkartenbasierte Simulation von tröpfchenförmigen Flüssigkeiten auf Oberflächen. In *Schumann, Marco (Hrsg.) u.a.; Gesellschaft für Informatik, GI-Fachgruppe Virtuelle Realität und Augmented Reality: 5. Workshop der GI-Fachgruppe VR/AR*, pages 149–156, Aachen, 2008. Shaker.

[19] Patrick Riess and Yvonne Jung. Module- and chain-based architecture for creating virtual and augmented reality manuals. In *Schumann, Marco (Hrsg.) u.a.; Gesellschaft für Informatik, GI-Fachgruppe Virtuelle Realität und Augmented Reality: 5. Workshop der GI-Fachgruppe VR/AR*, pages 197–208, Aachen, 2008. Shaker.

[20] Yvonne Jung, Jens Keil, Harald Wuest, Timo Engelke, Patrick Riess, and Johannes Behr. Knowledge at your fingertips – Multi-touch interaction for GIS and architectural design review applications. In *GRAPP 2009: Proceedings of the 4th International Conference on Computer Graphics Theory and Applications*, pages 387–392. INSTICC Press, 2009.

[21] Yvonne Jung and Johannes Behr. Simplifying the integration of virtual humans into dialog-like VR systems. In *Proceedings of IEEE Virtual Reality 2009 Workshop: 2nd Workshop on Software Engineering and Architectures for Realtime Interactive Systems*, pages 41–50. Shaker, 2009.

[22] Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. X3DOM – a DOM-based HTML5/ X3D integration model. In Stephen Spencer, editor, *Proceedings*

Web3D 2009: 14th International Conference on 3D Web Technology, pages 127–135, New York, USA, 2009. ACM Press. (Honorable Mentions).

[23] Yvonne Jung and Johannes Behr. Gpu-based real-time on-surface droplet flow in x3d. In Stephen Spencer, editor, *Proceedings Web3D 2009: 14th International Conference on 3D Web Technology*, pages 51–54, New York, USA, 2009. ACM Press.

[24] Yvonne Jung and Johannes Behr. Towards a new camera model for x3d. In Stephen Spencer, editor, *Proceedings Web3D 2009: 14th International Conference on 3D Web Technology*, pages 79–82, New York, USA, 2009. ACM Press.

[25] Richard M. Beales, Ajay Chakravarthy, Rolf Hedtke, Wolfgang Huther, Christoph Jung, Yvonne Jung, Stefanos Koutsoutos, and Angelos Yannopoulos. Automated 3d pre-vis for modern production. In *International Broadcasting Convention (IBC) 2009. Conference Publication: Technical Papers*, London, 2009. 8 p.

[26] Yvonne Jung, Christine Weber, Jens Keil, and Tobias Franke. Real-time rendering of skin changes caused by emotions. In Zsófia Ruttkay, Michael Kipp, Anton Nijholt, and Hannes Högni Vilhjálmsson, editors, *Intelligent Virtual Agents, 9th International Conference, IVA 2009, Amsterdam, The Netherlands, Proceedings*, volume 5773 of *Lecture Notes in Artificial Intelligence*, pages 504–505, Heidelberg, 2009. Springer.

[27] Christine Weber and Yvonne Jung. Entwurf und Evaluation eines Modells zur Darstellung emotional verursachter Hautveränderungen. In *Gerndt, Andreas (Hrsg.) u.a.; Gesellschaft für Informatik, GI-Fachgruppe Virtuelle und Erweiterte Realität: 6. Workshop der GI-Fachgruppe VR/AR*, pages 245–256, Aachen, 2009. Shaker.

[28] Yvonne Jung and Sebastian Wagner. Emotional factors in face rendering. In *IADIS Multi Conference on Computer Science and Information Systems 2010: Proceedings IADIS Interfaces and Human Computer Interaction (IHCI)*, pages 354–358. IADIS Press, 2010.

[29] Johannes Behr, Yvonne Jung, Jens Keil, Timm Drevensek, Peter Eschler, Michael Zöllner, and Dieter W. Fellner. A scalable architecture for the HTML5/ X3D integration model X3DOM. In Stephen Spencer, editor, *Proceedings Web3D 2010: 15th International Conference on 3D Web Technology*, pages 185–193, New York, USA, 2010. ACM Press.

[30] Yvonne Jung, Sebastian Wagner, Johannes Behr, Christoph Jung, and Dieter W. Fellner. Storyboarding and pre-visualization with x3d. In Stephen Spencer, editor, *Proceedings Web3D 2010: 15th International Conference on 3D Web Technology*, pages 73–81, New York, USA, 2010. ACM Press.

[31] Yvonne Jung, Sabine Webel, Manuel Olbrich, Timm Drevensek, Tobias Franke, Marcus Roth, and Dieter W. Fellner. Interactive textures as spatial user interfaces in x3d. In Stephen Spencer, editor, *Proceedings Web3D 2010: 15th International Conference on 3D Web Technology*, pages 147–150, New York, USA, 2010. ACM Press.

[32] Karsten Schwenk, Yvonne Jung, Johannes Behr, and Dieter W. Fellner. A modern declarative surface shader for x3d. In Stephen Spencer, editor, *Proceedings Web3D 2010: 15th International Conference on 3D Web Technology*, pages 7–15, New York, USA, 2010. ACM Press.

[33] Ajay Chakravarthy, Richard Beales, Yvonne Jung, Sebastian Wagner, Christoph Jung, Angelos Yannopoulos, Stefanos Koutsoutos, Rolf Schiffmann, Rolf Hedtke,

and Ignace Saenen. A notation based approach to film pre-vis. In *7th European Conference on Visual Media Production (CVMP 2010)*, pages 58–63, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[34] Rolf Hedtke, Rolf Schiffmann, Christoph Jung, Yvonne Jung, Ajay Chakravarthy, Richard Beales, Stefanos Koutsoutos, and Angelos Yannopoulos. Automatische 3D Visualisierung für die Film- und Fernsehproduktion. *FKT – Fachzeitschrift für Fernsehen, Film und elektronische Medien (FKTG)*, 64(12):645–652, 2010.

[35] Phillip Slusallek, Johannes Behr, Yvonne Jung, and Kristian Sons. Browser(t)räume; X3DOM & XML3D: Deklaratives 3D in HTML5. *iX – Magazin für professionelle Informationstechnik*, 11:54–63, 2010.

[36] Phillip Slusallek, Johannes Behr, Yvonne Jung, and Kristian Sons. Erdverbunden; X3DOM & XML3D: Transformationen und Interaktion. *iX – Magazin für professionelle Informationstechnik*, 12:116–121, 2010.

[37] Yvonne Jung, Johannes Behr, and Holger Graf. X3dom as carrier of the virtual heritage. In *3D-ARCH 2011: 3D Virtual Reconstruction and Visualization of Complex Architectures (4th Intl. Workshop)*. To appear.

[38] Yvonne Jung, Holger Graf, Johannes Behr, and Arjan Kuijper. Mesh deformations in X3D via CUDA with freeform deformation lattices. In *HCI International 2011*. Springer. To appear.

[39] Yvonne Jung, Arjan Kuijper, Dieter W. Fellner, Michael Kipp, Jan Miksatko, Jonathan Gratch, and Daniel Thalmann. Believable Virtual Characters in Human-Computer Dialogs. In *Eurographics 2011 – State of the Art Reports*. Eurographics Association. To appear.

Curriculum Vitae

Personal Data

Name	Yvonne Alexandra Jung
Address	Birkenhof, 55496 Argenthal, Germany
Nationality	German

Education and Work Experience

Since 10/2010	Research associate in the newly founded Visual Computing System Technologies group at Fraunhofer IGD in Darmstadt
01/'04 – 09/'10	Research associate in the Virtual and Augmented Reality group at the Fraunhofer Institute for Computer Graphics Research (IGD) in Darmstadt, Germany
06–12/2003	Research assistant at ZGDV e.V. in Darmstadt, Germany
2000 – 2002	Student assistant (tutor for programming in C/C++) at FH Bingen
06/2003	Graduation as Diplom-Informatikerin (FH) with major in software engineering at FH Bingen Topic of diploma thesis: "Entwurf einer virtuellen Lernwelt"
08–12/2002	Internship at Fraunhofer IGD in Darmstadt – topic of corresponding term paper: "Visualisierung von Konturlinien und Isoflächen in hierarchischen Gittern"
1999 – 2003	Studies: Applied Computer Science at the University of Applied Sciences (FH Bingen) in Bingen, Germany Topic of student research project: "Algorithmen zur Modellierung und Visualisierung von 3D-Objekten"
Summer 1999	Internship at prosozial gmbh in Halsenbach, Germany